11027-6003-R0-00

SYSTEMS IMPROVED NUMERICAL DIFFERENCING ANALYZER

USERS MANUAL

GASKI, J.D., FINK, L.C., ISHIMOTO, T.

September, 1970

NASA Contract 9-8289

TRW SYSTEMS
ONE SPACE PARK • REDONDO BEACH, CALIFORNIA

TRW SYSTEMS
REDONDO BEACH, CALIFORNIA

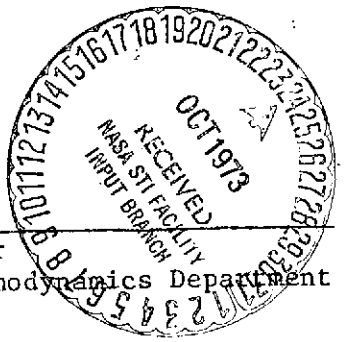
Prepared for
National Aeronautics and Space Administration
Manned Spacecraft Center
Under Contract NASA 9-8289

Prepared by:

J. D. Gaski

L. C. Fink

T. Ishimoto



Approved by:

J. T. Bevans, Manager
Heat Transfer & Thermodynamics Department

Approved by:

R. L. Dotts
NASA Technical Monitor
NASA Manned Spacecraft Center

PREFACE

Major improvements and numerous additions to the CINDA-3G program were generated under NASA Contract NAS 9-8289, "Development of Digital Computer Program for Thermal Network Correction". The improved program has been given the acronym SINDA (Systems Improved Numerical Differencing Analyzer) not only to reflect the major changes that have been made but also to indicate the inherent capabilities of the program.

This SINDA User's Manual generated under the NASA contract cited above necessarily draws heavily from the CINDA-3G User's Manual; SINDA has been programmed to accept the input data of CINDA-3G. Major additions that are described herein are concerned with sensitivity analysis and thermal network correction.

A particular note of interest to users is the semi-annual short course entitled "Workshop in Heat Transfer Computer Programs" offered by the University of California at Los Angeles Extension. This course which has provided numerous "workshops" on the use of CINDA-3G in the past several years will be updated to reflect the present SINDA program.

The monitoring of this NASA program was provided by Mr. R. Dotts; his helpful suggestions and forthright critiques are gratefully acknowledged. The authors are also indebted to Mrs. Dorothy Gramlich for suggestions on the document organizations and the typing of this manuscript.

TABLE OF CONTENTS

TABLE OF CONTENTS (Cont.)

TABLE OF CONTENTS (Cont.)

1. INTRODUCTION

1.1 Background

The original CINDA[*1] (Chrysler Improved Numerical Differencing Analyzer) computer program which was developed by the Thermodynamics Section of the Aerospace Physics Branch of Chrysler Corporation Space Division at NASA Michoud Assembly Facility was coded in FORTRAN-II and FAP for the IBM-7094 computers. CINDA was the result of an intensive analytical, engineering and programming effort. Numerous thermal analyzer-type programs were surveyed and several were studied in-depth. The foundation for CINDA was the storage and addressing of only the information required for the network solution and the systems features which allowed the reutilization of core storage area and brought into core only those instructions necessary for the solution of a particular problem. A systems compiler computer program that automatically optimized the utilization of computer core space was developed. This meant the generation of an integrated operation of relative addressing, packing features, peripheral tape storage units and overlay features.

CINDA evolved into CINDA-3G[2] which was developed by the same group that generated CINDA with a major portion of the work done under contract NASA/MSC NAS9-7043. CINDA-3G was essentially rewritten in order to take advantage of the improved systems software and machine speeds of the 3rd generation computers. CINDA was unsuitable for standard operation on third generation computers; it was virtually a self contained program having its own Update, Monitor and Compiler. On the other hand, CINDA-3G consisted of a preprocessor (written in FORTRAN) which accepted the user input data and converted it into advanced FORTRAN language subroutines and block data input which was then passed onto the system FORTRAN Compiler. This required a double pass on data where previously only one was required but the increased speed and improved software of the third generation machines more than compensated for the double pass.

1.2 SINDA

SINDA (Systems Improved Numerical Differencing Analyzer) was developed by the Heat Transfer and Thermodynamics Department of TRW Systems Group. The majority of the improvements and subroutine additions to CINDA-3G was done as part of the NASA/MSC contract NAS 9-8289 entitled "Development of Digital Computer Program for Thermal Network Correction." Programming and systems integration were directed to the UNIVAC-1108 computer.**

---

* Superscript numbers refer to the references in the Reference Section.

** The UNIVAC-1108 computer at the Jacobi Computation Center, Santa Monica, California was used in this study in order to insure operation under the 65K and 131K versions of the EXEC-II operating system

SINDA relied quite heavily on CINDA-3G and data deck compatibility has been rigorously followed; CINDA-3G data decks should be directly operational on the SINDA program. The primary differences between SINDA and CINDA-3G are: (1) elimination wherever possible of assembly language coding; (2) increased mnemonic options to aid the program user in data input; (3) inclusion of a second pseudo computer sequence for evaluation of nonlinear network elements; and (4) additional subroutines such as STEP (sensitivity analysis) and KALØBS-KALFIL (Kalman filtering).

SINDA program options offer the user a variety of methods for solution of thermal analog models presented in a network format. The network represents a one-to-one correspondence to both the physical and mathematical models. This analogy facilitates the construction of mathematical models of complex thermophysical systems and the preparation of program input. SINDA contains numerous subroutines for handling interrelated complex phenomena such as sublimation, diffuse radiation within an enclosure, simultaneous 1-D incompressible fluid flow including valving and transport delay effects, etc. The optional combination of these capabilities available in SINDA in conjunction with allowable large model size (greater than 4000 nodes for a linear 3-D system on 65K core) provides the user with a versatile analytical tool.

2. MNEMONICS AND NOTATION

This section was generated to assist the user in identifying the numerous mnemonic codes and some of the more commonly used notation. Note that the mnemonic codes that contain interpolation or polynominal are in terms of temperature except as noted.

2.1 Mnemonics

Code		Page
BCD	Binary Coded Decimal	4-1
BIV	Bivariate Interpolation Variable	4-10,4-18
	Replaces old DEC code	4-1 ,4-7,4-14
CAL	CALculate	4-7,4-14
CGD	Code used in CINDA-3G (has been replaced by	4-7
	DIV but will be accepted by SINDA)	4-9,4-16
CGS	Code used in CINDA-3G (has been replaced by	
	SIV but will be accepted by SINDA)	4-8, 4-12,4-15
DIM	Double Interpolation Multiple	4-9,4-17
DIT	Double Interpolation with Time as Variable	4-12
DIV	Double Interpolation Variable (replaces CGD	
	of CINDA-3G)	4-9,4-16
DPM	Double Polynomial Multiple	4-9,4-18
DPV	Double Polynomial Variable	4-9, 4-17
DTV	Double Interpolation with Time and temperature	
	as Variables	4-13
END	END of a block of input	4-2
GEN	GENerate	4-8, 4-11,4-15
LPCS	Long Pseudo Compute Sequence	3-10
ØCT	OCTal word	4-2
REM	Serves same function as FØRTRAN comment card	4-2
SIM	Single Interpolation Multiple	4-8,4-16
SIT	Single Interpolation with Time as Variable	4-12
SIV	Single Interpolation Variable (replaces CGS of	
	CINDA-3G)	4-8, 4-12,4-15
SPCS	Short Pseudo Compute Sequence	3-10
SPM	Single Polynomial Multiple	4-9,4-17
SPV	Single Polynomial Variable	4-9,4-17

2.2 Notation

A	Array address Area
C	Nodal capacitance ($=\rho VC_p$)
C_p	Specific Heat
F	Multiplying factor
G	Conductor for linear temperature difference Represents the radiation coefficient for fourth power temperature difference
G#	Conductor number
IG	Increment for the generated conductors
IN	Node generation increment
INA, INB	Increment for the generated adjoining nodes
k	Thermal conductivity
K#	Address of a constant's location
NA, NB	Adjoining node numbers
N#	Node number
#G	Number of conductors
#N	Number of nodes
t	Time
T	Temperature
Ti	Initial Temperature
V	Volume
W	Factor
x	Coordinate
X	Factor
y	Coordinate
Y	Factor
z	Coordinate
Z	Factor
α	Thermal diffusivity
ρ	Density

3. METHOD OF FINITE DIFFERENCE AND SINDA PROGRAM CONSIDERATIONS

3.1 Method of Finite Difference

3.1.1 Lumped-Parameter Representation

The key to utilizing a network type analysis program lies in the user's ability to develop a lumped parameter representation of the physical problem.[3] Once this is done, superposition of the network mesh is a mechanical task at most and the numbering of the network elements is simple although perhaps tedious. It might be said that the network representation is a "crutch" for the engineer, but, it does simplify the data logistics and allow easy preparation of data input to the program. In addition, it allows the user to uniquely identify any element in the network and modify its value or function during the analysis as well as sense any potential or current flow in the network. Another feature of the network is that it has a one-to-one correspondence to the mathematical model as well as the physical model.

The following diagram displays the lumped parameter representation and network superposition of a one dimensional heat transfer problem.

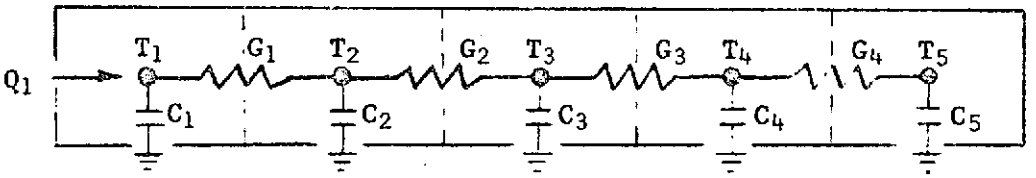


Figure (3-1)

The "node" points are centrally located within the lumps, and temperatures T at the nodes are considered uniform throughout the lump. The capacitors C from the nodes indicate the ability of the lump to store thermal energy. Capacitance values are calculated as lump volume times density times specific heat. The conductors (electrical symbol G) represent the capability for transmitting thermal energy from one lump to another. Conductor values for energy transmission through solids are calculated as thermal conductivity times the energy cross sectional flow area divided by path length (distance between nodes). Conductor values for convective heat transfer are calculated as the convection coefficient times the energy cross sectional flow area. Conductors representing energy transfer by radiation are usually indicated by crossed arrows over the conductor symbol. Radiation is nonlinear; it is proportional to the difference of the absolute temperatures raised to the fourth power. Utilization of the Farenheit system allows easy automation of this nonlinear transfer function by the program and reduces the input radiation conductor value to the product of the Stefan-Boltzmann constant times the surface area times the net radiant interchange factor (script F).[4,5]

Perhaps the most critical aspect of the lumped parameter approach is determining the lump size. There are methods for optimizing the lump size but they usually involve more analytical effort and computer time than the original analysis. One must also keep in mind that for a transient problem, time is being lumped as well as space. Of prime importance is what information is being sought from the analysis. If spot temperatures are being sought, nodes must at least fall on the spots and not include much more physically than would be expected to exist at a relatively similar temperature. Nodes must fall at end points when a temperature gradient is sought. Of necessity, lumping must be fairly fine where isotherms are sought. Lumping should be coarse in areas of high thermal conductivity. When nonlinear properties are being evaluated the lumping should be fine enough so that extreme gradients are not encountered. The lumping is also dependent on the severity of the nonlinearity.

In order to reduce round-off error the explicit stability criteria of the lump (the capacitance value divided by the summation of conductor values into the node) should be held fairly constant. This value ($C/\Sigma G$) is directly proportional to the square of the distance between nodes. Although refining the lumped parameter representation will yield more accurate answers, halving the distance between nodes decreases the stability criteria by a factor of four and increases the number of nodes by a factor of two, four or eight depending upon whether the problem is one, two or three dimensional. For the explicit case, halving the distance between nodes increases the machine time for transient analysis by a factor of eight, sixteen or thirty-two respectively. The increase in solution time for the implicit methods is somewhat less but proportional.

When lumping the time space, consideration must be given to the frequency of the boundary conditions. A time step must not step over boundary excitation points or they will be missed. Do not step over pulses, rather, rise and fall with them. Generally the computation interval for the explicit methods is sufficiently small so that frequency effects can be ignored. However, care must be exercised when specifying the time step for implicit methods. If only a small portion of a transient analysis involves frequency considerations the time step used may be selectively restricted for that interval. By setting the maximum time step allowed as a function of time, an interpolation call may be utilized to vary it accordingly.

One must also realize that the problem being solved is linearized over the time step. Heating rate calculations are usually computed for a time point and then applied to a time space. If the rates are nonlinear a certain amount of error is introduced, particularly so with radiation. These nonlinear effects may cause almost any method of solution to diverge. A brute force method for forcing convergence is to limit the temperature change allowed over the time space. Consideration of the factors mentioned above, coupled

with some experience in using the program, will aid the observant analyst in choosing lump sizes that will yield answers of sufficient engineering accuracy with a reasonable amount of computer time.[4,5]

3.1.2 Basics of Finite Differencing

The concept of network superposition on the lumped parameter representation of a physical system is easy to grasp. Describing the network to the program is also quite straightforward. Having described a network to the program, what information have we really supplied and what does the program do with it? Basically, we desire the solution to a simultaneous set of partial differential equations of the diffusion type; i.e.,

$$\frac{\partial T}{\partial t} = \alpha \nabla^2 T + S \quad , \quad \nabla^2 \equiv \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} + \frac{\partial^2}{\partial z^2} \tag{3-1}$$

That the diffusivity ($\alpha = k/\rho C_p$) may be temperature varying or nonlinear radiation transfer occurring is immaterial at this point. Of importance is how equation (3-1) is finite differenced and its relationship to the network and energy flow equations more commonly utilized by the engineer. The partial of the T state variable with respect to time is finite differenced across the time space as follows:

$$\frac{\partial T}{\partial t} \simeq \frac{T' - T}{\Delta t} \tag{3-2}$$

where the prime indicates the new T value after passage of the Δt time step.

The right side of equation (3-1) could be written with T primed to indicate implicit "backward" differencing or unprimed to indicate explicit "forward' differencing. The following equation is illustrative of how "backward" and "forward" combinations may be obtained.[6]

$$\frac{\partial T}{\partial t} = \beta(\alpha \nabla^2 T + S) + (1 - \beta)\ (\alpha' \nabla^2 T' + S') \tag{3-3}$$

$$0 \leq \beta \leq 1$$

Any value of β less than one yields an implicit set of equations which must be solved in a simultaneous manner (more than one unknown exists in each equation). Any value of β equal to or less than one half yields an unconditionally stable set of equations, or in other words, any time step desired may be used. Values of β greater than one half invoke stability criteria or limitations on the magnitude of the time step. A value of β equal to one half yields an unconditionally stable implicit set of equations commonly known as "forward-backward" differencing or the Crank-Nicholson method.[7] Various transformations or first order

integration applied to equation (3-1) generally yield an implicit set of equations similar to equation (3-3) with β equal to one half. The following finite difference approach generally applies to transformed equations.

Let's consider the right side of equation (3-3) with β equal to one and rewrite it as follows:

$$\alpha V^2 T+S \simeq \frac{\alpha}{\Delta x}\left(\frac{\partial T}{\partial x-}-\frac{\partial T}{\partial x+}\right)+\frac{\alpha}{\Delta y}\left(\frac{\partial T}{\partial y-}-\frac{\partial T}{\partial y+}\right)+\frac{\alpha}{\Delta z}\left(\frac{\partial T}{\partial z-}-\frac{\partial T}{\partial z+}\right)+S \qquad (3\text{-}4)$$

The minus or plus signs on the first partial denominator terms indicate that they are taken on the negative or positive side respectively of the point under consideration and always in the same direction. If we consider three consecutive points (1, 2 and 3) ascending in the x direction we can complete the finite difference of the x portion of equation (3-4) as follows:

$$\frac{\alpha}{\Delta x}\left(\frac{\partial T_2}{\partial x-}-\frac{\partial T_2}{\partial x+}\right)\simeq\frac{\alpha}{\Delta x}\left(\frac{T_1-T_2}{\Delta x-}+\frac{T_3-T_2}{\Delta x+}\right) \qquad (3\text{-}5)$$

Applying the above step to the y and z portions of the equation (3-4) yields the common denominator of volume ($V = \Delta x*\Delta y*\Delta z$). Using equation (3-3) with β equal to one, finite differencing with the steps used for equations (3-3), (3-4), and (3-5), substituting $\alpha = k/\rho C_p$ and multiplying both sides by $\rho V C_p$ yields:

$$\begin{aligned}\frac{\rho V C_p}{\Delta t}(T_o' - T_o) = {} & \frac{kAx}{\Delta x-}(T_1 - T_o)+\frac{kAx}{\Delta x+}(T_2 - T_o)\\ & +\frac{kAy}{\Delta y-}(T_3 - T_o)+\frac{kAy}{\Delta y+}(T_4 - T_o)\\ & +\frac{kAz}{\Delta z-}(T_5 - T_o)+\frac{kAz}{\Delta z+}(T_6 - T_o)+Q \qquad (3\text{-}6)\end{aligned}$$

where,

$Ax = \Delta y*\Delta z$

$Ay = \Delta x*\Delta z$

$Az = \Delta x*\Delta y$ and

$Q = \rho V C_p S$

The numbering system corresponds to the following portion of a three dimensional network (Figure 3-2).

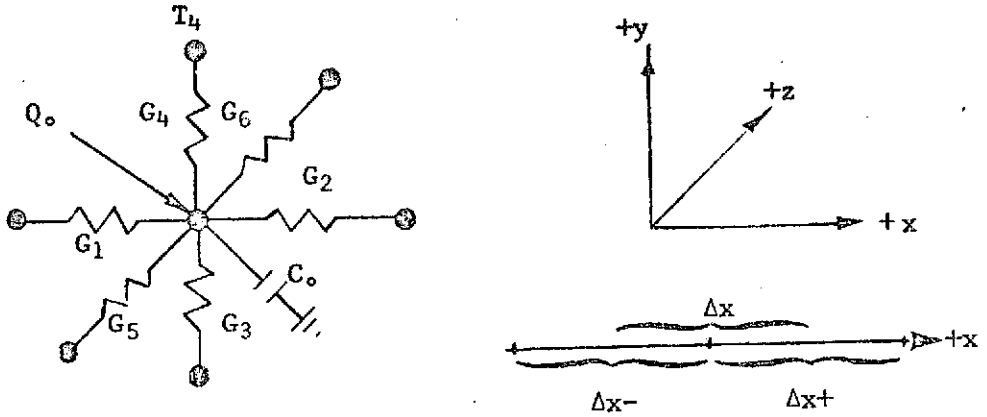


Figure (3-2)

It should be obvious that the network capacitance value is ρVC_p, that the G1 value is kAx/Δx-, etc. Equation (3-6) may be written as

$$C_o(T_o'-T_o)/\Delta t = G_1(T_1-T_o) + G_2(T_2-T_o) = G_3(T_3-T_o) + G_4(T_4-T_o) + G_5(T_5-T_o) + G_6(T_6-T_o) + Q_o \quad (3\text{-}7)$$

or in engineering terminology the rate of change of temperature with respect to time is proportional to the summation of heat flows into the node.

It should be noted that Figure (3-2) is essentially superpositioned on a lumped parameter cube of a physical system and is the network representation of equation (3-1). Since equation (3-7) is written in explicit form, only one unknown (T_o') exists and all of the information necessary for its solution is contained in the network description. If it had been formulated implicitly it would have to be solved in a simultaneous manner. No matter what method of solution is requested of the program, the information necessary has been conveyed by the network description. When an implicit set is used with β greater than zero, the energy flows based on old temperatures are added to the Q term and the equations are then treated in the same manner as for β equal to zero.

$$\alpha\nabla^2 T + S = 0 \quad (3\text{-}8)$$

The solution of Poisson's equation (3-8) is the solution utilized for steady state analysis. It is extremely important because all of the unconditionally stable implicit methods reduce to it. If equation (3-7) had all the right side values primed and the left side was subtracted from both sides, we could think of Co/Δt as a Go term and T_o (old) would then become a boundary node. In a manner of speaking, the capacitor we look at in 3-D becomes a conductor in 4-D.

We could draw a four dimensional network but since there is no feedback in time it is senseless to take more than one time step at a time. However, various time-space transformations can be utilized such that a one-dimensional "transient" analysis yields the solution to a two dimensional steady state problem, etc. This is analogous to the "Particle in Cell" method developed in the nuclear field for following shock wave propagation.

3.1.3 Iterative Techniques

Now that we have discussed the correlation between the physical model, network model and mathematical model, let's investigate the commonality of the various methods of solution. By describing the network of Figure (3-1) to the program we have supplied it with five temperatures, five capacitors, five sources (four not specified and therefore zero), four conductors and the adjoining node numbers of the conductors. An explicit formulation such as equation (3-6) has only one unknown. Its solution is easily obtainable as long as any associated stability criteria are continuously satisfied. A more interesting formulation would be a set of implicit equations as follows:

$$
\begin{aligned}
(T_1' - T_1)C_1/\Delta t &= Q_1' + G_1(T_2' - T_1') \\
(T_2' - T_2)C_2/\Delta t &= Q_2' + G_1(T_1' - T_2') + G_2(T_3' - T_2') \\
(T_3' - T_3)C_3/\Delta t &= Q_3' + G_2(T_2' - T_3') + G_3(T_4' - T_3') \\
(T_4' - T_4)C_4/\Delta t &= Q_4' + G_3(T_3' - T_4') + G_4(T_5' - T_4') \\
(T_5' - T_5)C_5/\Delta t &= Q_5' + G_4(T_4' - T_5')
\end{aligned}
\qquad (3\text{-}9)
$$

If the above had been formulated as a combination of explicit and implicit, the known explicit portion would have been calculated and added to the Q terms, then the β factor divided into the Q terms and multiplied times the Δt term.

If we divide the Δt term into the C terms and indicate this by priming C we can reformulate (3-9) as follows:

$$
\begin{aligned}
(C_1' + G_1)\quad T_1' &= Q_1' + C_1'T_1 + G_1T_2' \\
(C_2' + G_1 + G_2)T_2' &= Q_2' + C_2'T_2 + G_1T_1' + G_2T_3' \\
(C_3' + G_2 + G_3)T_3' &= Q_3' + C_3'T_3 + G_2T_2' + G_3T_4' \\
(C_4' + G_3 + G_4)T_4' &= Q_4' + C_4'T_4 + G_3T_3' + G_4T_5' \\
(C_5' + G_4)\quad\quad &= Q_5' + C_5'T_5 + G_4T_4'
\end{aligned}
\qquad (3\text{-}10)
$$

This equation can be generalized as:

$$
T_i' = \frac{C_i'T_i + \Sigma G_a T_a' + Q_i'}{C_i' + \Sigma G_a} \qquad (3\text{-}11)
$$

where the sub a indicates connection to adjoining nodes. A C' value of zero yields the standard steady state equation, the conductor weighted mean of all the surrounding nodes. We see here that the C' can be thought of as a conductor to the old temperature value and therefore equation (3-11), although utilized to obtain transient solutions, can be considered as a steady state equation in 4-D. By rewriting equations (3-10) in the form of equation (3-11) we are in a position to discuss iterative techniques. By assuming all old values on the right hand side of (3-10) we could calculate a new set of temperatures on the left which, although wrong, are closer to the correct answer. This single set of calculations is termed an iteration. By replacing all of the old temperatures with those just calculated we can perform another iteration. This process is called "block" iteration. A faster method is to utilize only one location for each temperature. This way, the newest temperature available is always utilized, otherwise old. This method is termed "successive point" iteration and is generally 25% faster than "block" iteration. The iterative process is continued a fixed (set by user) number of times or until the maximum absolute difference between the new and old temperature values is less than some prespecified value (set by user).

Although the above operations are similar to a relaxation procedure there is a slight difference. We are performing a set of calculations in a fixed sequence. A relaxation procedure would continuously seek the node with the maximum temperature difference between old and new and calculate it. Programming wise, as much work is required in the seeking operation which must be consecutive as in the calculation. For this reason it would be wasteful to code a true relaxation method.

In addition to the iterative approach, several solution subroutines utilize an acceleration feature and/or a different convergence criteria. Once it can be determined that the temperatures are approaching the steady state value, an extrapolation is applied in an attempt to accelerate convergence. This convergence criteria is the maximum absolute temperature change allowed between iterations. This criteria however is generally one sided and any associated errors are accumulative. In order to obtain greater accuracy, some subroutines are coded to perform an energy balance on the entire system (a type of Green's function) and apply successively more severe convergence criteria until the system energy balance (energy in minus energy out) is within some prespecified tolerance.

3.2 SINDA Program Considerations

3.2.1 Systems Programming

SINDA is more an operating system rather than an applications program. The more one studies and uses the program the more apparent this becomes. In order for the program to accomplish the desired operations with regard to overlay features, data packing, dynamic storage allocation, subroutine library file and yet be written in Fortran, it was necessary to program SINDA as a preprocessor. This preprocessor operates in an integral fashion with a large library of assorted subroutines which can be called in any sequence desired yet operate in an integrated manner. It reads all of the input data, assigns relative numbers, packs them, forms the pseudo-compute sequences and writes the operations blocks on a peripheral unit as Fortran source language with all of the data values dimensioned exactly in name common. It then turns control over to the system Fortran compiler which compiles the constructed subroutines and enters execution. The Fortran allocator has access to the SINDA subroutine library and loads only those subroutines referred to by the problem being processed.

Due to this type of operation, SINDA is extremely dependent on the systems software supplied. However, once the program has been made operational on a particular machine, the problem data deck prepared by the user can be considered as machine independent. The user need only be aware of the control cards and deck setup requirements at his particular installation.

3.2.2 Pseudo-Compute Sequence

When working with a simultaneous set of equations such as equation (3-10), they are quite often treated by matrix methods and formulated as follows:

$$A \left\{T'\right\} = \left\{B\right\} \tag{3-12}$$

where

$$A = \begin{bmatrix} (C_1'+G_1) & -G_1 & 0 & 0 & 0 \\ -G_1 & (C_2'+G_1+G_2) & -G_2 & 0 & 0 \\ 0 & -G_2 & (C_3'+G_2+G_3) & -G_3 & 0 \\ 0 & 0 & -G_3 & (C_4'+G_3+G_4) & -G_4 \\ 0 & 0 & 0 & -G_4 & (C_5'+G_4) \end{bmatrix} \tag{3-13}$$

and

$$\{T'\} = \begin{Bmatrix} T_1' \\ T_2' \\ T_3' \\ T_4' \\ T_5' \end{Bmatrix} \qquad \{B\} = \begin{Bmatrix} Q_1'+C_1'T_1 \\ Q_2'+C_2'T_2 \\ Q_3'+C_3'T_3 \\ Q_4'+C_4'T_4 \\ Q_5'+C_5'T_5 \end{Bmatrix},$$

The inverse of [A] is then calculated and the solution obtained by matrix multiplication.

$$\{T'\} = [A]^{-1} \{B\} \tag{3-14}$$

It should be noted that the one dimensional problem has no more than three finite values in any row or column of the coefficient matrix [A]. A three dimensional problem would generally have no more than seven finite values in any row or column. It is easy to see that a one thousand node three dimensional problem would require one million data locations of which approximately 993,000 would contain zero. The inverse might require an additional one million data locations. Aside from exceeding computer core area, the computer time required to calculate the inverse is proportional to the cube of the problem size and large problems soon become uneconomical to solve.

The explicit and iterative implicit methods previously discussed are well suited for optimizing the data storage area required and reducing the solution time. Note the adjoining node numbers associated with the conductors of Figure (3-1) as shown in Table (3-1).

Table (3-1)

G# N# N#		
1,1,2	→	G1 between nodes 1 and 2
2,2,3	→	G2 between nodes 2 and 3
3,3,4	→	G3 between nodes 3 and 4
4,4,5	→	G4 between nodes 4 and 5

Note also the row and column position of conductor values off the main diagonal in the [A] coefficient matrix, equation (3-13). By retaining the adjoining node numbers for each conductor we are able to identify their element position in the coefficient matrix. As a consequence, we need store only the finite values. The main diagonal term is a composite of the node capacitance and conductor values off of the main diagonal.

The SINDA preprocessor operates on the adjoining node numbers to form what is termed a pseudo-compute sequence (PCS). The nodes are to be calculated sequentially in ascending relative order so the

conductor adjoining node numbers are searched until number one is found. When this occurs the conductor number and other adjoining node number are stored in a single core location. Several indicators are also stored in this single core location. They reveal if the capacitor of the node under consideration is non-linear, of a source from the source data block is present (required for thermal network correction) and whether the conductor value is nonlinear, radiation, one way or the last one to the node under consideration. The search is continued until all ones are located. The process is then continued for node two, etc. until all the node numbers have been processed. The pseudo-compute sequence formed is shown in Table (3-2). A slight variation to this operation is to place a minus sign on the original other adjoining node number so that it is not recognized when it is searched for. The resulting pseudo-compute sequence thus formed is shown in Table (3-3).

Table (3-2) Long Pseudo-Compute Sequence (LPCS)

last G	var C	var G	rad Q	G#	one way	N#	var Q
1				1		2	
				1		1	
1				2		3	
				2		2	
1				3		4	
				3		3	
1				4		5	
1				4		4	

Table (3-3) Short Pseudo-Compute Sequence (SPCS)

last G	var C	var G	rad Q	G#	one way	N#	var Q
1				1		2	
1				2		3	
1				3		4	
1				4		5	
1				0		0	

The above pseudo-compute sequences are termed long (LCPS) and short (SPCS) respectively. By starting at the top of the pseudo-compute sequence we are operating on node one. The G# and N# values identify the conductor into the node (the position of the conductor value in an array of conductor values) and the adjoining node (the position of the temperature, capacitor and source values in arrays of temperature, capacitor and source values respectively). The node being operated on starts as one and is advanced by one each time a last conductor indicator is passed.

It is easy to see that the long pseudo-compute sequence identifies the element position and value locations of all the off diagonal

elements of the row being operated on. It takes complete advantage of the sparsity of the coefficient matrix. It is well suited for "successive point" iteration of the implicit equations because all elements in a row are identified. When a row is processed and the new T value obtained, the new T can then be used in the calculation procedure of succeeding rows.

The short pseudo-compute sequence identifies each conductor only once and in this manner takes advantage of the symmetry of the coefficient matrix as well as the sparsity. It is well suited for explicit methods of solution. The node being operated on and the adjoining node number reveal their temperature value locations and their source value locations. The explicit solution subroutines calculate the energy flow through the conductor, add it to the source location of the node being worked on and subtract it from the source location for the adjoining node. However, if the short pseudo-compute sequence were utilized for implicit methods of solution they would require the use of slower "block" iterative procedures. The succeeding rows do not have all of the elements defined and the energy rates passed ahead were based on old temperature values.

The variable capacitor, conductor and source indicators in the above pseudo-compute sequences are no more than yes or no switches, each occupying one bit of the core location. Each time a variable switch is found yes a location counter is increased by one and used as a pointer to an entry point in a second pseudo-compute sequence. The location indicated contains three values in the core location; the type of variable or nonlinearity and the array and constant locations of required data for evaluating the function. This method of storing information on nonlinear network elements is extremely conservative of core space and also quite efficient.

3.2.3 Data Logistics

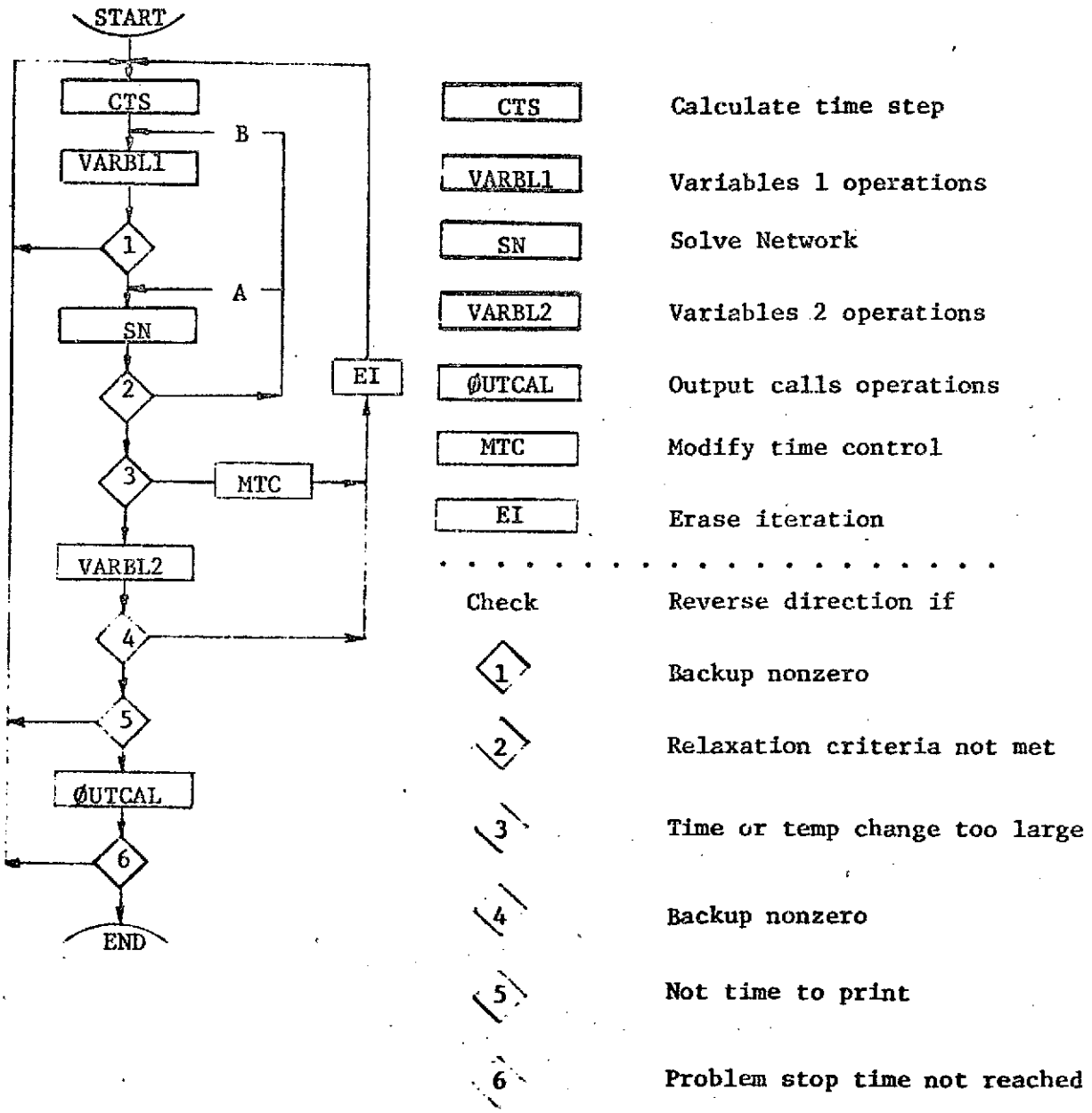
The long and short pseudo-compute sequences formulated as shown previously allow the program to store only the finite values in the coefficient matrix thereby taking advantage of its sparsity. In addition, the short pseudo-compute sequence takes advantage of any symmetry which may exist. Multiple connected conductors which will be covered in the next section also allow the user to take advantage of similarity as well. The foregoing is fairly easy to follow, especially if the nodes and conductors start with the number one and continue sequentially with no missing numbers. This restriction is too limiting for general use on large network models. To overcome this restriction the program assigns relative numbers (sequential and ascending) to the incoming node data, conductor data, constants data and array data in the order received. Any numbers missing in the actual numbering system set up by the user are packed out thereby requiring only as much core space as is actually necessary.

All network solution subroutines require three locations for diffusion node data (temperature, capacitance and source) and one location for each conductor value. They also may require from zero to three extra locations per node for intermediate data storage. Each node in a three dimensional network has essentially six conductors connected to it but only three are unique; i.e., each additional node requires only three more conductors. Hence, each node in a three dimensional system requires from six to nine storage locations for data values (temperature, capacitance, source, three conductors and up to three intermediate locations). The two integer values and six indicators that make up the first pseudo-compute sequence are packed into a single core location. Hence, for a three dimensional network, each node requires approximately three locations for data addressing for the short and six locations for the long pseudo-compute sequence. The number of core locations required per node can vary from nine to fifteen exclusive of the second pseudo-compute sequence for nonlinear elements.

The program requires the user to allocate an array of data locations to be used for intermediate data storage and initialize array start and length indicators. Each subroutine that requires intermediate storage area has access to this array and the start and length indicators. They check to see that there is sufficient space, update the start and length indicators and continue with their operations. If they call upon another subroutine requiring intermediate storage, the secondary subroutine repeats the check and update process. Whenever any subroutine terminates its operations it returns the start and length indicators to their entry values. This process is termed "Dynamic Storage Allocation" and allows subroutines to share a common working area.

3.2.4 Order of Computation

A problem data deck consists of four data and four operations "blocks" which are preprocessed by SINDA and passed on to the system FØRTRAN compiler. The operations blocks are named EXECUTIØN, VARIABLES 1, VARIABLES 2 and ØUTPUT CALLS. The SINDA preprocessor constructs these blocks into individual subroutines with the entry names EXECTN, VARBL1, VARBL2 and ØUTCAL respectively. After a successful FØRTRAN compilation, control is passed to the EXECTN subroutine. Therefore, the order of computation depends on the sequence of subroutine calls placed in the EXECUTIØN block by the program user. No other operations blocks are performed unless called upon by the user either directly by name or indirectly from some subroutine which internally calls upon them. The network solution subroutines listed in Section A.2 internally call upon VARBL1, VARBL2 and ØUTCAL. Their internal order of computation is quite similar, the primary difference being the analytical method by which they solve the network. Figure (3-3) represents a flow diagram of all the network solution subroutines; the subroutine writeups contain the comparisons made at the various check points and routings taken.



BASIC FLOW CHART FOR NETWORK SOLUTION SUBROUTINES

FIGURE (3-3)

4. DATA INPUT REQUIREMENTS

4.1 General

A SINDA problem data deck consists of both data and instruction cards. The card reading subroutines for SINDA do not utilize a fixed format type of input; they use a free form format quite similar to the old SHARE decimal data read routine. The type of data is designated by a mnemonic code in columns eight, nine and ten. This is followed by the data field which consists of columns twelve through eighty or the instruction field which consists of columns twelve through seventy-two. Although blanks are allowed before or after numerical data, they may not be contained within; that is, the number 1.234 is fine, but 1. 234 will cause the program to abort. The program processes and stores the problem data as FØRTRAN name common data and reforms instructions into FØRTRAN source language which are then passed on to the system FØRTRAN compiler. Instruction cards which contain an F in column one are passed on exactly as received, except that the F is repositioned to column 80. Cards containing a C in column one are passed on as received to become FØRTRAN comment cards. Any instruction card with or without an F in column one may contain a statement or sequence number in columns two through five which is passed on to and used by the FØRTRAN compiler.

4.1.1 Mnemonic Codes

4.1.1.1 Old DEC Code (replaced by three blanks) and Dollar Sign ($)

The most frequently used mnemonic code was the old DEC designation which has been replaced by three blanks. The data following this blank mnemonic code may be one or more integers, floating point numbers (with or without the E exponent designation) or alpha-numeric words of up to six characters each. The reading of a word or number continues until a comma is encountered and then the next word or number is read. As many numbers or words as desired may be placed on a card but they may not be broken between cards. A new card is equivalent to starting with a comma and therefore no continuation designation is required or allowed. All blanks are ignored and reading continues until the terminal column is reached or a dollar sign encountered. Comments pertinent to a data card may be placed after a dollar sign and are not processed by the program. If sequential commas are encountered, floating point zero values are placed between them.

4.1.1.2 BCD Code

The next most frequently used code is BCD (for binary coded decimal) which must be followed by an integer one through nine in column twelve. The integer designates the number of six character words immediately following it. Blanks are retained and only the designated number of six character words are read from the card.

4.1.1.3 END

Mnemonic code END is utilized to designate the end of a block of input to the program.

4.1.1.4 REM

Code REM serves the same function as a FØRTRAN comment card; it is not processed by the program but allows the user to insert non-data for clarification purposes.

4.1.1.5 Codes for Nonlinear Elements

Special codes for generation and/or evaluation of nonlinear elements are discussed in a later paragraph.

4.2 Input Blocks

The data deck prepared by a program user consists of various input "blocks" containing either data or instructions. There are either two or four data blocks (an additional one is optional) and four operations blocks in addition to the title block. A fixed sequence of block input as indicated below is required and each block must start with a BCD 3 header card and terminate with an END (mnemonic codes). At the end of the deck that contains the data and operations blocks, a termination card (BCD 3END ØF DATA) must be used. Note that even though an input block is not required for a given problem, all input blocks must be input, source excepted.

Before presenting details about the various data and operations blocks, it may be helpful if a list of required input blocks with a brief description of each were presented at this time.

(1) Title Block (Refer to Section 4.2.1)

(Col) 8 12

BCD 3GENERAL
or BCD 3THERMAL SPCS
or BCD 3THERMAL LPCS

Comment: The TITLE block normally contains a header generated by by the user. The GENERAL indicates that the problem is non-network and thus requires no node or conductor data block. The THERMAL cards indicate that the problem is represented as a network and that either a short (SPCS) or long (LPCS) pseudo-compute sequence is to be constructed (Paragraph 3.2.2). Specification of LPCS or SPCS is obtained from the execution subroutines (Section A.2). Note that LPCS subroutines must not be mixed with the SPCS subroutines. An END (mnemonic code) is required.

(2) Node Data Block (Refer to Section 4.2.2)

(Col) 8 12

BCD 3NØDE DATA

Comment: The NODE DATA block contains the node number, the type (diffusion, arithmetic, and boundary), the initial temperature, and the capacitance if applicable.

A number of options that are concerned with the sequential generation of nodes, temperature varying capacitance, etc. is available for specific requirements.

This data block is not required if the title block is BCD 3GENERAL.

An END (mnemonic code) card is required.

(3) Optional Source Data Block (Refer to Section 4.2.3)

(Col) 8 12

BCD 3SØURCE DATA

Comment: Optional means that if data for the Q block is not an input, then the block header card and the END (mnemonic code) need not be included in the data deck.

The SØURCE DATA block contains the node number and the source value.

A number of options that involve temperature varying sources, time varying sources, sequential generation of sources. etc. is available for specific requirements.

An END (mnemonic code) card is required

(4) Conductor Data Block (Refer to Section 4.2.4)

(Col) 8 12

BCD 3CØNDUCTØR DATA

Comment: The CØNDUCTØR DATA block contains the conductor number, the type (linear or radiation), adjoining node number, and conductor values.

A number of options for specific requirements, such as the sequential generation of conductors, temperature varying conductors, etc., is available.

This data block is not required if the title block is BCD 3GENERAL.

An END (mnemonic code) card is required.

(5) Constants Data Block (Refer to Section 4.2.5)

(Col) 8 12

BCD 3CØNSTANTS DATA

Comment: The CØNSTANTS DATA block is always inputted as doublets. The doublet may be a control constant and value or user constant and value. User constants are simply data storage locations or control constants having alphanumeric names and the values are communicated through programs common to specific subroutines which require them.

(6) Array Data Block (Refer to Section 4.2.6)

(Col) 8 12

BCD 3ARRAY DATA (Refer to Section 4.2.6)

Comment: The ARRAY DATA input consists of an array number, a sequential list of information and termination with an END (data END, not mnemonic).

An END (mnemonic code) card is required.

(7) Execution Operations Block (Refer to Section 4.2.7 and Paragraph 4.2.7.2)

(Col) 8 12

BCD 3EXECUTIØN

Comment: The EXECUTIØN operations block is the first of four operations blocks (EXECUTIØN, VARIABLES 1, VARIABLES 2, and ØUTPUT CALLS). These four operations blocks are preprocessed by SINDA and passed on to the system FØRTRAN compiler as four separate subroutines, EXECTN, VARBL1, VARBL2, and ØUTCAL.

None of the operations specified in VARBL1, VARBL2, or ØUTCAL will be performed unless called either directly by name in the EXECUTION block or internally by a subroutine.

An END (mnemonic code) card is required.

(8) Variables 1 Operation Block (Refer to Section 4.2.7 and Paragraph 4.2.7.3)

(Col) 8 12

BCD 3VARIABLES 1

Comment: The VARIABLES 1 operations block allows a user pre-solution operations. Thus the user may specify the network (evaluation of nonlinear network elements, coefficients and boundary values) prior to entering the network solution phase.

An END (mnemonic code) card is required.

(9) Variables 2 Operations Block (Refer to Section 4.2.7 and Paragraph 4.2.7.4)

(Col) 8 12

BCD 3VARIABLES 2

Comment: The VARIABLES 2 operations block allows the user to perform post-solution operations. That is the solved network may be examined for quantities such as nodal heat flow, compare calculated values with test data, etc.

An END (mnemonic code) card is required.

(10) Output Calls Operations Block (Refer to Section 4.2.7 and Paragraph 4.2.7.5)

(Col) 8 12

BCD 3ØUTPUT CALLS

Comment: The ØUTPUT CALLS operations block allows a user to call upon any desired subroutine with its contents printed in the output interval. Several subroutines for printing output and plotting are available.

An END (mnemonic code) card is required.

(11) End of Data (Refer to Paragraph 4.2.7)

(Col) 8 12

BCD 3END ØF DATA

Comment: Input blocks (1) -(10) above must be terminated by the END ØF DATA card.

(12) Parameter Runs (Refer to Paragraph 4.2.8)

(Col) 8 12

BCD 3INITIAL PARAMETERS
or BCD 3FINAL PARAMETERS

Comment: Parametric analysis which does not involve network or operation changes may be performed on the same computer run. Only data values such as output page heading, temperatures, capacitances, conductances, arrays and constants may be changed.

The parameter run decks are inserted in the problem data deck immediately preceding the BCD 3END ØF DATA card.

(13) Store and Recall Problem Options (Refer to Paragraph 4.2.9)

Comment: The store and recall capacity allows an indefinite time lapse between parametric analysis; the store subroutine call may be used as many times as desired. The recall is activated by a single card that replaces the blank card (refer to Appendix E) that precedes the problem data deck and must be followed by initial parameter and block data change cards exactly as shown for parameter runs, including the first BCD 3 parameter and End Cards and the BCD 3END ØF DATA card.

4.2.1 Title Block

The first card of a problem data deck is the title block header card. It conveys information to the program as to the type of problem, which data blocks to follow and how they should be processed. The three options presently available are:

(Col) 8 12

BCD 3GENERAL
or BCD 3THERMAL SPCS
or BCD 3THERMAL LPCS

The GENERAL indicates that a non-network problem follows and therefore no node or conductor data is present. The THERMAL cards indicate that a conductor-capacitor (CG) network description follows and that either a short (SPCS) or long (LPCS) pseudo-compute sequence should be constructed. The title block header card may be followed by as many BCD cards as desired. However, the first twenty words (six characters each) are retained by the program and used as a page heading by the user designated output routines. The block must be terminated by an END card and is then followed by node data for a CG network problem or constants data for a non-network problem.

4.2.2 Node Data Block

4.2.2.1 Definition and Designation

There are three types of nodes, diffusion, arithmetic and boundary. All nodes are renumbered sequentially (from one on) in the group order received. The user input number is termed the actual number, while the program assigned number is termed the relative node number. This relative numbering system allows sequential packing of the data and does not require a sequential numbering system on the part of the program user. It is worth noting that the pseudo compute sequence is based on the relative numbering system; this means that the computational sequence of nodes is identical with their group input sequence. If a user desires to reorder the computations in order to aid boundary propagation, it is necessary to reorder only the nodal input data.

The user may intermix the three types of nodes; the SINDA preprocessor sorts the nodes into the three basic groups in order to conserve core space.

Diffusion Nodes

Diffusion nodes are those nodes with a positive capacitance and thus store energy. In these nodes, temperatures are calculated by using a finite difference representation of the parabolic differential equation. A diffusion node causes three core locations to be reserved, one each for temperature, capacitance, and a source.

Arithmetic Nodes

Arithmetic nodes have no capacitance and are designated by a negative capacitance value. Temperatures of these nodes are calculated by a finite difference representation of Poisson's partial differential equation. This is a steady state calculation that always utilizes the latest diffusion node values available. Arithemtic nodes reserve only temperature and source locations.

Boundary Nodes

Boundary nodes are designated by a minus sign on the node number; these nodes reflect mathematical boundaries not necessarily the physical boundary. Boundary temperatures are not changed by the network solution subroutines, but may be modified as desired by the user. A boundary node receives only a temperature location.

4.2.2.2 Mnemonic Codes for Node Data

Several mnemonic codes are available including the generation and/or evaluation of nonlinear network elements; under node data, capacitance is the network element.

Standard Input for a Node (Three Blank Mnemonic Code)

Node data input with the three blank mnemonic code always consists of three values; the integer node number followed by the floating point initial temperature and capacitance values. A negative capacitance value is used to designate an arithmetic node, while a negative node number designates a boundary node. Although the capacitance value of a boundary node is meaningless, it must be included in order to maintain the triplet formed.

(Col) 8 12

```
N#, Ti,C
4 ,70.,1.3          (example 1)
5 ,70.,-1.0         (example 2)
-6,70.,1.0          (example 3)
```

where, N# represents the node number (always an integer)
Ti represents the initial temperature
C represents the nodal capacitance

The example 1 indicates a diffusion node number 4 with a temperature of 70. degrees and capacitance of 1.3; example 2 indicates an arithmetic node 5 with a temperature of 70. degrees and a capacitance of -1.(any negative number could have been used); example 3 represents a boundary node 6 with an arbitrary capacitance of 1.0.

CAL Option

This option allows the SINDA user to input the nodal capacitance as a composite; the capacitance C is calculated as X times Y times Z times W.

(Col) 8 12

```
CAL N#, Ti, X, Y, Z,W
CAL 10,80.,1.,2.,3.,4.          (example)
```

where, N# is the node number
Ti is the initial temperature
X,Y,Z, & W are factors

The example shows node 10 with a temperature of 80. degrees and a capacitance of 24.(1. x 2. x 3. x 4.).

GEN Option

The GEN option allows the user to generate a sequence of nodes.

(Col) 8 12

```
GEN  N#, #N, IN, Ti ,C
GEN  6 , 3 , 2 , 75.,10.
```

where, N# is the starting node number (integer)
#N is the number of nodes to be generated (integer)
IN is the node generation increment (integer)

The example given generates a sequence of 3 nodes 6, 8, and 10 all at 75. degrees and a capacitance of 10. As a note of interest, the user may input the capacitance value as the X,Y,Z and W composite shown for the CAL Option.

SIV Option (Identical to the CINDA-3G CGS which is SINDA acceptable)

The SIV option allows the user to specify a temperature varying capacitance.

(Col) 8 12

```
SIV N#, Ti, A, F
SIV 5 ,80.,A1,2.4          (example 1)
SIV 5 ,80.,A1,K7           (example 2)
```

where, A represents the array address of a doublet array to be linearly interpolated with the node temperature as the independent variable.
F represents a multiplying factor for the capacitance; it may be a literal (actual value as shown in example 1) or the address of a constant's location containing the actual value (example 2).

SIM Option

This is a combination of the GEN and SIV options; notation and description follows directly from the previous presentation.

(Col) 8 12

```
SIM N#,N#,IN, Ti, A,F
SIM 3 , 2, 5,80.,A1,4.2          (example)
```

The example given will generate nodes 3 and 8 (both at 80. degrees) and with a linearly interpolated temperature varying capacitance which is multiplied by 4.2 . Capacitance value is calculated as density, ρ, times specific heat, C_p, times volume, V, $C = \rho V C_p$. If both ρ and C_p are temperature varying, the user must reference an array of $\rho * C_p$ versus T with the multiplication factor being V.

25<

DIV Option (Identical to the CINDA-3G CGD which is SINDA acceptable)

The DIV option allows the user to calculate the temperature varying capacitance of a node consisting of two dissimilar materials. It can be thought of as two SIV calls with the result added to the nodal capacitance.

(Col) 8 12

```
DIV N#, Ti, A1, F1, A2, F2
DIV 5 ,80.,A14,2.4,A15,5.3          (example 1)
DIV 5 ,80.,A14,2.4,1.0,5.3          (example 2)
DIV 5 ,80.,1.0, K3,A15, K4          (example 3)
```

Example 1 shows both capacitances as temperature varying; examples 2 and 3 show several ways of inputting when only one of the capacitances is time varying. Note that the constant capacitance is calculated in example 2 as 1.0 x 5.3 and in example 3 as 1.0 x value in K3.

DIM Option

The DIM option is a combination of the GEN and DIV options and its operation follows the description of the GEN and DIV options.

(Col) 8 12

```
DIM N#,#N,IN, Ti, A1, F1, A2, F2
DIM 4 ,3 ,2 ,80.,A14,4.2,A17,K36          (example)
```

The example generates nodes 4, 6 and 8, all at 80. degrees and with a composite capacitance of 4.2 times the value interpolated from A14 added to the product of K36 times the value interpolated from A17.

SPV, SPM, DPV and DPM Options

These options are identical to the options, SIV , SIM , DIV , and DIM respectively, with the exception that the arrays referenced contain polynomial coefficients for evaluation of the temperature varying capacitance.

(Col) 8 12

```
SPV N#, Ti, A, F
SPM N#, #N,IN, A, F
DPV N#, Ti,A1,F1,A2,F2
DPM N#, #N,IN,A1,F1,A2,F2
SPV 5 ,80.,A4,7.6                    (example)
```

The example is for the SPV option; node 5 is at 80. degrees and the capacitance is evaluated from the polynomial coefficients in array 4, the temperature of node 5 and multiplied by 7.6. If array 4 had the following input,

4,2.3,0.8,.006,1.2E-5,END

capacitance C5 would be calculated as:

$$C5 = 7.6*(2.3+0.8*T5+0.006*T5^2+1.2E-5*T5^3)$$

The largest order of the polynomial that the program can accommodate is estimated to be eight.

BIV Option

The BIV option allows the user to specify a bivariate capacitance of a node. The nodal temperature is the X independent variable and time is the Y independent variable. Time as used here is the mean time for the iteration and is obtained internally with TIMEM as the control constant which will be discussed in a later section called Constants Data Block.

(Col) 8 12

BIV N#,Ti,A,F (Refer to page A.4-12 for form of A)

Example of a Node Data Block

A node data block utilizing the preceding mnemonic options is listed below as an example, which does not correspond to a particular problem, but merely illustrates the data input format. It should be noted that the types of nodes may be intermixed (diffusion, arithmetic and boundary) and that two sets of mnemonic cards may be on the same card. Caution: the data for a node must be on a single input card.

(Col) 8 12

```
BCD 3NØDE DATA
    1,80.,1.2,2,80.,1.3          $ two diffusion nodes (old DEC)
CAL 3,80.,1.,2.,3.,4.            $ one      "      node
GEN 4,2,1,80.,2.7                $ two      "      nodes
GEN 6,2,1,80.,-1.0               $  "   arithmetic  "
GEN -8,2,1,-460.,1.0             $  "   boundary nodes
SIV 10,80.,A1,4.63,11,80.,A1,2.5 $ two single material nodes
SIM 12,2,1,80.,A1,3.25           $  "      "       "        "
DIV 14,80.,A1,2.31,A2,K5         $ one double     "      node
DIM 15,2,1,80.,A1,K4,A2,2.8      $ two    "       "      nodes
SPV 17,80.,A3,1.8                $ one single     "      node
SPM 18,3,1,80.,A3,2.3            $ three  "       "      nodes
DPV 21,80.,A3,1.4,A4,1.8         $ one double     "      node
DPM 23,2,180.,A3,0.4,A4,2.9      $ two    "       "      nodes
BIV 25,80.,A5,4.76               $ bivariate capacitance
END
```

4.2.3 Optional Source Data Block

4.2.3.1 Definition

Optional means that if there are no data for the source data block then the block header card and the END (mnemonic code) need not be included in the data deck. As in the node data block, the user input number is the actual number and the program assigned number is the relative source number. Within the optional source data block, the source may be a constant, a function of time, a function of temperature, or a function of both time and temperature. It should be noted that a source may not be impressed on a boundary node.

4.2.3.2 Mnemonic Codes for Source Data

Several mnemonic codes are available including the generation of time and temperature varying sources.

Standard Input for a Source (Three Blank Mnemonic Code)

Source data input with the three-blank mnemonic code consists of the node number and a constant value which may be either a user constant or a literal.

(Col) 8 12

```
N#, Q
3,2.3                    (example 1)
5, K2                    (example 2)
```

where, N# represents the node number (always an integer)
Q represents either a user constant or a literal (a literal in example 1 and a user constant in example 2)

GEN Option

The GEN option allows the user to impress the same heat source on a number of equally incremented nodes.

(Col) 8 12

```
GEN N#, #N, IN, Q
GEN 7, 3, 2, 4.3         (example 1)
GEN 7, 3, 2, K2          (example 2)
```

where, N# is the starting node number
#N is the number of nodes
IN is the node generation increment

The examples given will generate a sequence of three nodes, 7, 9, and 11 all with an impressed source of 4.3 (example 1) or with an actual value in constants location K2 (example 2).

SIV Option

The SIV option allows the user to specify a temperature varying source.

(Col) 8 12

```
SIV N#, A, F
SIV 9, A2, 5.3                (example 1)
SIV 9, A2, K3                 (example 2)
```

where, A represents the array address of a doublet array to be linearly interpolated with the node temperature as the independent variable.
F represents a multiplying factor for the source; F may be a literal (actual value of 5.3 as shown in example 1) or the address of a constants location K3 containing the actual value (example 2).

SIT Option

The SIT option allows the user to specify a time varying source.

(Col) 8 12

```
SIT N#, A, F
SIT 9, A2, 5.3                (example 1)
SIT 9, A2, K3                 (example 2)
```

where, A represents the array address of a doublet array to be linearly interpolated with time as the independent variable (TIMEM).
F represents a multiplying factor for the source; it may be a literal (actual value of 5.3 as shown in example 1) or as the address of a constants location K3 containing the actual value (example 2).

DIT Option

The DIT options allow the user to specify two time varying sources that are a function of time. That is the total heat into node i is represented as,

$$Q_i(t) = k_1 f_1(t) + k_2 f_2(t)$$

where, k_1 and k_2 are constants
$f_1(t)$ and $f_2(t)$ are functions of time (TIMEM).

(Col) 8 12

```
DIT N#, A1, K1, A2, K2
DIT 6, A3, K7, A4, K3         (example 1)
DIT 5, 2.4, K6, A3, K7        (example 2)
DIT 5, 2.4, 6.2, A3, K7       (example 3)
DIT 6, 2.4, 6.2, A3, 3.7      (example 4)
```

where, A1 and A2 are arrays (A1 and A2 may be literals but not simultaneously).
K1 and K2 may be user constants or literals.

Example 1 means that the heat into node 6 is the sum of the interpolated value of array three times the actual value in constants address K7 plus the interpolated value of array four times the actual value in constants location K3. Example 2 shows array A1 to be a literal, 2.4 whereas both A1 and K1 are literals in example 3. In example 4, A1, K1, and K2 are literals.

DTV Option

The DTV option allows a user to specify a heat source that is both time and temperature dependent. The heat into node i is,

$$Q_2 = k_1 f_1(t) + k_2 f_2(T)$$

where, k_1 and k_2 are constants
$f_1(t)$ is a function of time (TIMEM).
$f_2(T)$ is a function with temperature as a variable

(Col) 8 12

```
DTV N#, A1, K1, A2, K2
DTV 7, A3, K7, A4, K3          (example 1)
DTV 6, 2.4, K6, A3, K7         (example 2)
DTV 6, 2.4, 6.2, A3, K7        (example 3)
DTV 6, 2.4, 6.2, A3, 3.7       (example 4)
```

where, A1 and A2 are arrays (A1 and A2 may be literals but not simultaneously).
K1 and K2 may be user constants or literals.

Example 1 means that the heat into node 7 is the sum of the interpolated value of array three times the actual value in constants address K7 plus the interpolated value of array four times the actual value in constants location K3. Example 2 shows array A1 to 6 literal with a value of 2.4. Example 3 shows both A1 and K1 to be literals and in example 4, A1, K1, and K3 to be literals.

4.2.4 Conductor Data Block

4.2.4.1 Definition

Two basic types of conductors may be used, regular or radiation; either may utilize temperature varying properties in calculating the conductance value. It should be noted that the regular conductor is associated with the linear temperature difference, $T_i - T_j$; as a result the regular conductor input and output has the dimensions of a conductor. On the other hand, the radiation conductor is

inputted as σb_{ij} of the radiation term $\sigma b_{ij}(T_i^4-T_j^4)$

where, σ is the Stefan-Boltzmann constant (.1714 x 10^{-8} Btu/hr $ft^2{}^\circ R^4$)
b_{ij} is a radiation coefficient that includes shape factor and properties

The radiation conductor printout is σb_{ij} .

4.2.4.2 Mnemonic Codes

The mnemonic codes discussed under node data are available under conductor data with slightly revised meanings.

Blank Mnemonic Code (Standard Conductor Input)

When utilizing the blank mnemonic code a regular conductor consists of the integer conductor number followed by two integer adjoining node numbers and the floating point conductance value. If more than one conductor has the same constant value, these conductors may share the same conductor number and value. This is accomplished by placing two or more pairs of integer adjoining node numbers between the conductor number and the value.

(Col) 8 12

```
G#,NA,NB,G
1 ,1 ,2 ,2.3                          (example 1)
2 ,2 ,3 ,  3, 4, 4, 5,7.6             (example 2)
-3,4 ,9 ,1.8 E-10                     (example 3)
4,-5,6,4.3                            (example 4)
```

where, G# stands for the integer conductor number
NA and NB are adjoining node numbers
G is the conductor value (for linear temperature difference) and represents the radiation coefficient for fourth power temperature difference.

Example 1 is a regular conductor number 1, between nodes 1 and 2 with a value of 2.3. Example 2 demonstrates the use of multiple connections and can be used only for constant conductors; node number 2 has a value of 7.6 and is used between nodes 2 & 3, 3 & 4, and 4 & 5. Example 3 shows a radiation coefficient number 3 between nodes 4 & 9 with a value of 1.8E-10. Example 4 illustrates the use of a one-way conductor (refer to the GEN option below for details).

CAL Option

The CAL option for the conductor data differs from the node data in that the conductor value is calculated as X times Y times Z divided by W. When using the X,Y,Z, & W input under the CAL option no addresses are allowed; all values must be floating point numbers.

(Col) 8 12

```
CAL G#,NA,NB, X, Y, Z,W
CAL 4 , 5, 6,1.,2.,3.,4.          (example)
```

The example conductor 4 between nodes 5 & 6 receives the value of 1.5.

GEN Option

The GEN option allows the user to generate a sequence of conductors and the increment values may be zero or negative. Inputs X,Y,Z,W under the CAL option may also be used with the GEN option. An additional feature of the program is the one way conductor which allows its effect (value) to be included in the calculation procedure of one adjoining node but not the other. One way conduction is indicated by placing a minus sign on the node number that does not include the one way conductor on its calculation. One way conductors may be used with any of the mnemonic options.

(Col) 8 12

```
GEN G#,#G,IG,NA,INA,NB,INB,G
GEN 5 , 3, 1, 1,  1,90, -1,4.7                          (example 1)

GEN G#,#G,IG,NA,INA,NB,INB,  X,  Y,     Z,W
GEN -8, 3, 0, 1,  1,99,  0,4.0,0.8,1.E-10,1.0           (example 2)
GEN 9 , 3, 0,-32, 1,33,  1,7.8                          (example 3)
```

where, #G stands for the number of conductors to be generated
IG, INA, & INB values are the incremental adjustments to the conductors, and the adjoining nodes.

Example 1 is for a regular conductor, example 2 for a radiation coefficient and example 3 is for a one way conductor. These examples are equivalent to the following:

```
5,1,90,4.7,6,2,89,4.7,7,3,88,4.7          (for example 1)
-8,1,99,2,99,3,99,3.2E-10                 (for example 2)
9,-32,33,-33,34,-34,35,7.8                (for example 3)
```

SIV Option (Identical to the CINDA-3G CGS which is SINDA acceptable)

The SIV option for conductor data allows linear interpolation of a temperature varying property. the interpolated value is then multiplied by the factor F to obtain the element value. If only one temperature is to be used for interpolation, the node (with the temperature to be used) is listed first and the factor F is set negative.

(Col) 8 12

```
SIV G# ,NA,NB, A, F
SIV 10 , 8, 9,A1,4.7                      (example 1)
SIV -11, 9,10,A2,-3.4E-10                 (example 2)
```

32<

In example 1, conductor 10 is evaluated with the arithmetic mean of temperatures 8 & 9 used as the independent variable in array A1 and 4.7 as the factor. Example 2 illustrates the case of a single temperature (number 9) used in the interpolation of array 2 and for a radiation coefficient.

SIM Option

The SIM mnemonic option is a combination of the GEN and SIV options.

(Col) 8 12

```
SIM G#,#G,IG,NA,INA,NB,INB, A,F
SIM 12, 3, 1, 7,  1,15,  2,A4,4.6                    (example)
```

The above example will generate three separate conductors, each of which are temperature varying and dependent upon the mean of the adjoining temperatures as follows:

12,7,15,A4,4.6,13,8,17,A4,4.6,14,9,19,A4,4.6

If the factor F (4.6) had been negative, the first nodes (7, 8, & 9) would have been used for the interpolation.

DIV Option (Identical to the CINDA-3G CGD which is SINDA acceptable)

The DIV option allows simulation of a conductor consisting of two serial dissimilar materials, one or both of which may be temperature varying. Two separate conductance values are computed and then summed as series conductors (regular conductors) or multiplied for effective conductance (radiation) That is, if G1 represents the regular conductance of one material and G2 the other, then the combined conductance (for series conductors is evaluated as:

$$G_{combined} = \frac{1}{\frac{1}{G1} + \frac{1}{G2}}$$

If G1 represents the emissivity $\epsilon 1$ of one surface and G2 contains the emissivity $\epsilon 2$ of the second surface, the combined radiation coefficient is evaluated as:

$$G_{combined} = G1*G2 \equiv \epsilon 1 * \epsilon 2 * F \text{ (F is a factor between surfaces 1 \& 2)}$$

If one of the two materials is not temperature varying, a literal is used in place of the array address and no interpolation is performed; the conductance is evaluated as the literal times the F value.

(Col) 8 12

```
DIV G# ,NA,NB,A1,    F1,A2,F2
DIV 15 ,12,14,A1,   2.3,A2,K7                        (example 1)
DIV -16,17,28,A4,2.E-10,A5,1.0                       (example 2)
```

In example 1 for regular conductor 15, temperature 12 is used with array 1 and the factor 2.3 to obtain the G1 value and temperature 14 is used with array 2 and the contents of K7 to obtain the GR value. Example 2 is for the radiation coefficient 16 with two temperature varying emissivities.

DIM Option

The DIM option is a combination of the GEN and DIV options. Either one or both of the two dissimilar materials may be temperature varying. The series conductance and product radiation coefficient calculation follow the method discussed under the DIV option.

(Col) 8 12

```
DIM G#,#G,IG,NA,INA,NB,INB,A1,  F1, A2,F2
DIM 17, 4, 1, 6,  1,16,  1,A4,16.6,3.4,7.2          (example)
```

In the example above, for the four regular conductors only one of the material is temperature varying and the other has a constant value (G2 = 3.4*7.2).

SPV Option

The SPV option is identical to the SIV option except that a polynomial solution is performed instead of interpolation. Either the temperature of the first nodal input or the mean temperature of the adjoining nodes is used for polynomial evaluation; the former is designated by a negative factor, -F, and the latter by a positive factor, F.

(Col) 8 12

```
SPV G#,NA,NB, A,F
SPV 21,32,42,A6,-4.3                    (example 1)
SPV 22,33,43,A6,3.8                     (example 2)
```

Example 1 uses the temperature of node 32 for interpolation, whereas in example 2 the mean temperature of nodes 33 and 43 is used for interpolation.

SPM Option

The SPM option is a combination of the GEN and SPV options. Directions for its use follow directly from the individual GEN and SPV descriptions.

(Col) 8 12

```
SPM G#,#G,IG,NA,INA,NB,INB, A,F
SPM 23, 2, 1,18,  1,99,  0,A6,K9                (example)
```

DPV Option

The DPV option is identical to the DIV option with the exception that the DPV option uses a polynomial evaluation in lieu of direct interpolation. One or both materials may have temperature varying properties.

ol) 8 12

```
DPV G#,NA,NB,A1, F1,A2,F2
DPV 25,27,28,A6,4.7,A7,2.3                    (example)
```

the example the conductances for the two materials are evaluated
parately and combined as series conductances or multiplied if radiation.

M Option

e DPM option is a simple combination of the GEN and DPV options. It is
entical to the DIM option except that polynomial evaluation is substituted
r linear interpolation.

ol) 8 12

```
DPM G#,#G,IG,NA,INA,NB,INB,A1,F1,A2,F2
DPM 26, 3, 1,29,  1,30,  1,A7,K4,A6,14.7      (example)
```

V Option

e BIV mnemonic option allows simulation of a bivariate property. The array
ferenced by the call must be a bivariate array where the X independent
riable is temperature and the Y independent variable is time. The mean
mperature of the adjoining nodes and the mean time (control constant TIMEM)
rough the program constants are used for interpolation. The result is then
ltiplied by the F factor to obtain the conductance value.

variate conductivity is generally encountered in superinsulations which are
bject to pressure changes. This often is due to convection effects when a
cuum is being pulled during test or during launch. Although the second
riable may be altitude or pressure, these variables can generally be
lated to time and thus simulated by use of the BIV option.

ol) 8 12

```
BIV G#,NA,NB, A,F
BIV 29,17,42,A9,7.8                           (example)
```

ogram Idiosyncrasy and Illustration of Conductor Input Options

A single valued conductor with as many adjoining node pairs as desired
y be used, extending several cards if necessary, however. In addition,
e mnemonic options may have more than one set of data on a card, but a
t of data may not be broken between cards.

Various conductor input options are illustrated below:

```
(Col)   8   12

      BCD 3CØNDUCTØR DATA
          1,1,2,1.2,2,2,3,1.7          $ two regular conductors
          3,3,4,4,5,5,6,1.5            $ triple placed conductor
          4,-7,8-8,9,7,6               $ double place one-way conductor
      CAL 5,4,5,1.4,3.7,2.6,8.2        $ calculated conductance
      GEN 6,3,1,6,1,6,1,4.8            $ generate three conductors
      SIV 9,10,11,A3,4.6               $ variable conductor, single
      SIM 10,2,1,11,1,12,1,A3,2.8      $ two variable conductors
      DIV 12,17,24,A3,4.1,A4,7.6       $ variable conductor, double
          -16,1,99,1.E-15              $ radiation conductor
      SPV 17,4,28,A5,13.7              $ variable conductor, single
      SPM -18,3,1,2,1,99,0,A5,1.4E-14  $ variable radiation conductor
      DPV 21,19,37,A5,4.3,A7,9.2       $ variable conductor, double
      DPM 22,4,1,20,138,1,A5,4.3,A7,10.6 $ four variable conductors
      BIV 29,29,43,A8,K4               $ bivariate conductor
      END
```

4.2.5 Constants Data block

Constants data are always input as doublets, the constant name or number followed by its value. They are divided into two types, control constants and user constants, and may be intermingled within the block.

4.2.5.1 User Constants

User constants, which are identified as numbers, are simply data storage locations which may contain integers, floating point numbers or up to six character alphanumeric words. The user must place data in user constant locations as needed and supply the location addresses to subroutines as arguments.

4.2.5.2 Control Constants

Control constants number forty-three and have alphanumeric names. Control constant values are communicated through program common to specific subroutines which require them. However, any control constant name desired can be used as a subroutine argument. Wherever possible, control constant values not specified are set to some acceptable value. If a required control constant value is not specified, an appropriate error message is printed and the program terminated. The user should check the description of subroutines being used to determine control constant requirements. A list of control constant names and brief description of each follows. Exact usage is found in the subroutine descriptions.

ARLXCA	The maximum arithmetic relaxation change allowed.
ARLXCC	The maximum arithmetic relaxation change calculated.
ATMPCA	The maximum arithmetic temperature change allowed.
ATMPCC	The maximum arithmetic temperature change calculated.

BACKUP	If non-zero, the completed time step is erased and repeated.
BALENG	User specified system energy balance to be maintained.
CSGFAC	Stability criteria multiplication/division factor.
CSGMAX	Maximum stability criteria for network. } (C/ΣG) max and min.
CSGMIN	Minimum stability criteria for netowrk. }
CSGRAL	Stability criteria range followed.
CSGRCL	Stability criteria range calculated.
DAMPA	Arithmetic node damping factor.
DAMPD	Diffusion node damping factor.
DRLXCA	The maximum diffusion relaxation change allowed.
DRLXCC	The maximum diffusion relaxation change calculated.
DTIMEH	Largest time step allowed (maximum).
DTIMEI	Input time step for implicit solutions.
DTIMEL	Smallest time step allowed (minimum).
DTIMEU	Time step used for all transient network problems.
DTMPCA	The maximum diffusion temperature change allowed.
DTMPCC	The maximum diffusion temperature change calculated.
ENGBAL	The calculated energy balance of the system.
LAXFAC	Linearization interval for subroutine CINDSM.
LINECT	A line counter location for program output.
LØØPCT	Program count of iteration loops performed (Integer).
NLØØP	User input number of iteration loops desired (Integer).
ØPEITR	Causes output each interation if set non-zero.
ØUTPUT	Time interval for activating ØUTPUT CALLS.
PAGECT	A page counter location for program output.
TIMEM	Mean time for the computation interval.
TIMEN	New time at the end of the computation interval.
TIMEND	Problem stop time for transient analysis.
TIMEØ	Old time at the start of the computation interval, also used as problem start time, may be negative.

ITEST,JTEST,KTEST,LTEST,MTEST

Dummy control constants with integer names.

RTEST,STEST,TTEST,UTEST,VTEST

Dummy control constants with non-integer names.

4.2.5.3 Example of Constants Data Block

The following is representative of a constants data block.

(Col) 8

```
BCD 3CØNSTANTS DATA
    TIMEND=10.0,ØUTPUT=1.0        $CØNTRØL CØNSTANTS
    1=10,2=3,3=7,4=8              $INTEGERS
    5=1.,6=1.E3,7=1.E-3           $FLØATING PØINT
    8=TEMP,9=ALPHA                $ALPHANUMERIC
END
```

SINDA will accept commas in place of the equal signs (indicated above) so as to remain compatible with CINDA-3G.

4.2.6 Array Data Block

4.2.6.1 Format

Array data input consists of an array number, a sequential list of information and termination with an END (data END, not mnemonic). For example,

(Col) 12

```
1,1.6,2.4,3.8,END
```

The example indicates array 1 with three data values.

4.2.6.2 Integer Count of Array Values

Numerous subroutines (interpolation, matrix, etc.) require that the exact number of values in an array be specified as an integer. In order to reduce the number of subroutine arguments and chance of error, the SINDA preprocessor counts the number of values in an array and supplies this integer count as the first value in the array. Subroutines whose array arguments require the array integer count will list the array argument as A(IC). Subroutines whose array arguments require the first data value rather than the integer count will list the array argument as A(DV).

Referring to the example of 4.2.6.1, by addressing A1 as a subroutine argument the integer count 3 would be the first value followed by 1.6, 2.4 and 3.8. If the user wanted the 1.6 data value to be addressed the argument should be A1+1.

4.2.6.3 Two Types of Alphanumeric Inputs and SPACE Option

One alphanumeric input allows each word to be separated by a comma, requires each word to start with a letter and does not allow the use of blanks. The other requires use of the BCD mnemonic code and the single integer word count (Col 12). It allows use of letters, numbers or characters anywhere and retains blanks. The SPACE option is an easy way for the user to specify a large number of locations which are initialized by the preprocessor as floating point zeros. The space option requires the word SPACE followed by the number of locations to be initialized. It may be used anywhere in an array and as many times as desired as long as total available core space is not exceeded. An example of these inputs is presented below.

(Col) 8 12

```
BCD 3ARRAY DATA
    1,1.6,2.4,3.8,END                          $FLØATING PØINT NUMBERS
    2,TEMP1,TEMP2,END                          $ALPHANUMERIC
    3                                          $ALPHANUMERIC
BCD 3TEMPERATURE STUDY
    END
    4,SPACE,100,END                            $SPACE ØPTIØNS
    5,4.7,2.3,SPACE,14,8.6,SPACE,17,END
END
```

4.2.7 Program Control

4.2.7.1 General Considerations

Data Check

The SINDA preprocessor does a significant amount of data checking as the data blocks (discussed in the previous paragraphs) are read. Detected input errors are noted and the card containing the error is identified. For example, when the adjoining node pairs under conductor data are read, an immediate check is made for these nodes as input under node data. Incorrect constraint names are also immediately identified.

Pseudo-Compute Sequence

After the four (or five) blocks have been read the preprocessor then forms the pseudo-compute sequences as described in Section 3.2.2.

Operations Blocks

Aside from the title block, there are either two or five data blocks depending upon whether the problem is GENERAL or THERMAL respectively. In either case, there are four operations blocks entitled EXECUTIØN, VARIABLES 1, VARIABLES 2 and ØUTPUT CALLS. The operations or instructions called for in these blocks determine the program control. They are preprocessed by SINDA and passed on to the system FØRTRAN compiler as four separate subroutines entitled EXECTN, VARBL1, VARBL2 and ØUTCAL respectively. When the FØRTRAN compilation is successfully completed, control is passed to the EXECTN subroutine which sequentially performs the operations in the same input order as specified by the user in the EXECUTIØN block. None of the operations specified in the other three blocks will be performed unless called either directly by name in the EXECUTIØN block or internally by a subroutine.

No operations will be performed unless requested by the user and no control constants will be utilized unless called upon by a subroutine. Network solution subroutines internally call upon VARBL1 VARBL2, and ØUTCAL (see Figure 3-3, page 3-13), and use numerous control constants. Details on these control constant requirements are presented in Section A.2. Network solution subroutines require no arguments but most others do. These arguments may be addresses which refer to the location of data or they may be literals; i.e, the actual data value. All of the input data can be addressed by using alphanumeric arguments of the following form.

TN for the temperature location of node N
CN for the capacitance location of node N
QN for the source location of node N
GN for the conductance location of conductor N
KN for the value location of constant N
AN for the starting location of array N
and control constants utilize their individual names.

Array Address

When addressing arrays the user must be careful to address correctly the integer count or the data value in the array (refer to Section 4.2.6). The user may also uniquely address any item in an array. For instance, the one hundredth value in array ten could be uniquely addressed as A10+100. This means of addressing is only available for arrays. If a user desired to address the twenty BCD words for the title block which were retained for output page headings, he could do so by using the argument H1, or any word individually, by Hn, n = 1,20.

Dynamic Storage Allocation

Dynamic Storage Allocation is a unique feature of the SINDA program. Although not carried to the ultimate, all subroutines which require working space generally obtain it from a common working array. However, the user must specify information about this array to the program. To do so the user must place three FØRTRAN cards at the start of the execution block; for example:

```
(Col) 1     7

      F     DIMENSIØN X(100)
      F     NDIM = 100
      F     NTH = 0
```

The names used must be exactly as shown; in the above example a working array of 100 locations is formed. If a different number of locations is needed the integer 100 is changed as desired (both first and second cards). If no working locations are required the cards may be omitted. The progrm user must check the writeups of subroutines he is using in order to determine if, when and how much of a working array is required.

An F in column one indicates to the preprocessor that the card is FØRTRAN and should be passed on as received. This F option allows the user to program FØRTRAN operations directly into the operations blocks. However, the SINDA arguments listed above are not FØRTRAN compatible with the exception of the control constant names. Therefore, it is recommended that the program user utilize SINDA subroutine calls wherever possible. This is impossible however when logical operations are required. In this case it is recommended that the user place SINDA data values as needed into the available dummy control constant names allowed for that purpose. Then, FØRTRAN logical operations can be utilized with the dummy control constant names as arguments. FØRTRAN statement numbers for routing purposes may be placed in columns two through five on any operations cards, either FØRTRAN or SINDA.

The data field for node, conductor, constant and array data consists of columns twelve through eighty. However, the data field

of operations cards ends with column seventy-two. In a manner of speaking, a SINDA subroutine call is a special array and should terminate with a data END. In order to simplify input for the user, the operations read subroutines recognize two special characters; the left and right parenthesis. The left parenthesis is accepted as a comma, while the right parenthesis is accepted as a comma followed by a data END. This allows what would have been:

```
(Col)       12
            ADD,K1,K2,K3,END
```

to be more esthetically formatted as:

```
            ADD(K1,K2,K3)
```

which is almost identical to a FØRTRAN subroutine call.

4.2.7.2 Execution Operations Block

An execution operation block might be as follows:

```
(Col) 1       8   12
             BCD 3EXECUTIØN
      F      DIMENSION  X(25)
      F      NDIM=25
      F      NTH=0
      F   10 TIMEND=TIMEND+1.0
                 CNFRWD              $EXPLICIT FØRWARD DIFFERENCING
                 STFSEP(T20,TTEST)   $PLACE T10 INTØ DUMMY CC
      F      IF(TTEST.LE.100.) GØ TØ 10
             END
```

The above indicates a transient thermal problem in which the user desires to terminate the analysis when the temperature at node 20 exceeds one hundred degrees. The problem must have been fairly small because only twenty-five working locations were dimensioned and CNFRWD requires one per node. It does demonstrate the use of both SINDA calls and FØRTRAN operations, and that control constants are referred to by name in either. Another example might be

```
(Col) 1       8   12

             BCD 3EXECUTIØN
      F      DIMENSIØN  X(500)
      F      NDIM=500
      F      NTH=0
                 CINDSL   $STEADY STATE (USES LPCS)
      F      TIMEND=10.0
                 CNFRWD   $TRANSIENT ANALYSIS (USES SPCS)
             END
```

In this case the user desires to have a steady state analysis performed on the network followed by a transient analysis utilizing the steady state answer as initial conditions. However, the two network solution subroutines that are cited are incompatible because CINDSL uses LPCS whereas CNFRWD uses SPCS; as a result the program would be terminated with an appropriate error message.

There is no end to the variety of examples that could be generated. In reality, the program user is actually programming although it is somewhat disguised as data input. However, the program does simplify the task of data logistics and automates overlay, tape library, and other systems features thereby greatly lessening the programming knowledge which might otherwise be required of a user.

A point well worth considering is proper initialization. All instructions contained in the other three operations blocks are performed each iteration or on the output interval. If an operation being performed in Variables 1 is utilizing and producing non changing constants, it should be placed in the Execution block (prior to the network solution call) so that it will be performed only once. Input arrays requiring post-interpolation multiplication for units conversion only could be prescaled, thereby deleting the multiplication process. Complex functions of a single independent variable requiring several interpolation values which are then combined in a multiplicative fashion can be precalculated versus the independent variable. Such a precalculated complex function reduces the amount of work performed during the transient analysis. A great many operations of this type can be performed in the Execution block prior to call for a transient analysis. Also, output operations to be performed once the transient analysis is completed may be placed directly into the Execution block following the transient network solution call.

4.2.7.3 Variables 1 Operations Block

The statement that this program solves nonlinear partial differential equations of the diffusion type is not quite accurate. In reality the program only solves linear equations. However, nonlinearities are evaluated at each computation interval and in this manner generally yield acceptable answers to nonlinear problems. This method is more properly termed quasilinearization. The Variables 1 operations block allows a point in the computational sequence at which the user can specify the evaluation of nonlinear network elements, coefficients and boundary values. The various mnemonic codes utilized for node and conductor data cause the construction of a pseudo-compute sequence which is used to evaluate nonlinearities. The user must specify any additional functions or nonlinearities as subroutine calls in Variables 1 in order to completely define the network prior to entering the network solution phase.

Prior to inclusion of the various mnemonic codes, the Variables 1 operations block primarily consisted of linear interpolation subroutine calls input by the user for the evaluation of temperature varying properties. While these linear interpolation calls are automated through use of the mnemonic codes, it is up to the program user to specify any required trivariate interpolations or other nonstandard functional evaluations necessary. Just prior to performing the Variables 1 operations, all network solution subroutines zero out all source locations. Therefore, the user is required to specify constant as well as variable or nonlinear impressed sources in this block if not specified in source data block. A Variables 1 operations block could be as follows:

```
(Col) 1       8   12
              BCD 3VARIABLES 1
                  STFSEP(10.0,Q17)                  $CØNSTANT IMPRESSED SØURCE
                  D1DEG1(TIMEM,A8,Q19)              $TIME VARYING SØURCE
                  D2D1WM(T18,TIMEM,A19,7.63,G18)    $BIVARIATE FUNCTIØN
      F       TTEST=11.6
      F       IF(TIMEN.GT.10.)  TTEST=0.0
                  STFSEP(TTEST,Q27)                 $VARIABLE SØURCE
              END
```

The first call above places a constant heating rate of 10.0 into the source location of node 17. The second call causes a linear interpolation to be performed on array 8 using mean time as the independent variable to obtain a time varying heating rate for node 19. The third call uses mean time and the temperature at node 18 as independent variables to perform a bivariate interpolation. The interpolated answer is then multiplied by 7.63 and placed as the conductance value of conductor 18. The next two cards are FØRTRAN and allow a value of 11.6 to be placed into control constant TTEST until TIMEN exceeds 10.00 after which a value of 0.0 is placed into TTEST. This amounts to a single step in a "stair-case" function. The last card places the value from TTEST into the source location for node 27. Another sample Variables 1 block might look as follows:

```
(Col) 1       8   12

              BCD 3VARIABLES 1
                  BLDARY(A12+1,T1,T7,T3,T4)     $CØNSTRUCT VECTØR
                  D1DEC1(T7,A19,A13+2)          $INTERPØLATION
                  IRRADE(A7,A13,A10,A12)        $IR RADIØSITY EXPLICIT
                  BRKARY(A12+1,Q1,Q7,Q3,Q4)     $DISTRIBUTE Q RATES
                  D1D1WM(TIMEM,A9,0.35,TTEST)   $INTERPØLATE
                  ADD(TTEST,Q1,Q1)              $ADD TWØ RATES
              END
```

The first call above causes the construction of an array of four temperature values necessary as input to an infrared radiosity subroutine (third card). The second call causes the linear

interpolation of a temperature varying property from array 19 to be placed into array 13+2 which is the second array argument for the radiosity call. This second argument must be an array of surface emissivities for the surfaces under consideration; therefore array 19 must be an array of temperature varying emissivity. The BRKARY call takes data values from array 12 + 1, 2, 3 and 4 and places them into the source locations for nodes 1, 7, 3 and 4 respectively. The fifth call performs linear interpolation on array 9 using TIMEM as the independent variable, multiplies the result by 0.35 and places it in control constant TTEST. This might be a time varying solar heating rate where 0.35 is the solar absorptivity. The ADD call adds this rate to what is already contained in the source location for node 1. Each node has one and only one source location. If a user desires to impress more than one heating rate on a node, he must sum the rates and supply the value to the single source location available per node.

The Variables 1 operations block is the logical point in the network computational sequence for the calculation of impressed sources whether they are due to internal dissipation of powered components, radiation depositation, aerodynamic heating or orbital heating. If a desired subroutine is not available, the user may always add his own; data communication is obtained through subroutine arguments as in any other subroutine.

4.2.7.4 Variables 2 Operations Block

In regards to the network solution, the Variables 1 operations may be thought of as pre-solution operations and Variables 2 operations as post-solution operations. In Variables 1 the network was completely defined with respect to nonlinear elements and boundary conditions. Variables 2 allows the user to look at the just solved network. He may meter and integrate flow rates, make corrections in order to account for material phase changes or compare just calculated answers with test data in order to derive empirical relationships. A simple Variables 2 operations block might be as follows:

```
(Col)   8    12

        BCD 3VARIABLES
            QMETER(T1,T2,G1,K1)        $METER HEAT FLØW
            QINTEG(K1,DTIMEU,K2)       $INTEGRATE HEAT FLØW
            RDTNQS(T5,T1,G8,K3)        $METER RADIATIØN FLØW
            QINTEG(K3,DTIMEU,K4)       $INTEGRATE RADIANT FLØW
            ADD(K2,K4,K5)
        END
```

The first call measures the heat flow from node one to node two through regular conductor one and stores the result in constant location one. The second call performs a simple integration with respect to time and sums the result into constants location two.

The third call measures heat flow through a radiation conductor which is then integrated by the fourth call. The sum of the two integrations is obtained by the fifth call. Another Variables 2 operations block might be as follows:

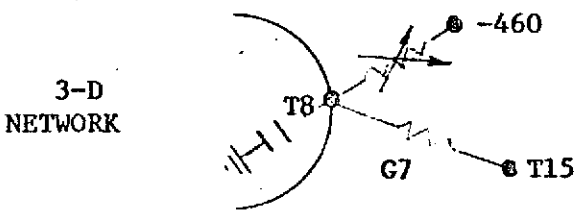
```
(Col)    8    12
         BCD 3VARIABLES 2
             ABLATS(A1,1.76,K8,A7,T15,C15)      $ABLATIVE ØN NØDE 15
         END
```

Phase change subroutines such as the above are unique in that they perform automatic corrector operations. Node 15 has been solved by the network solution subroutine as though no ablative existed. The ABLATS subroutine then corrects the temperature node 15 to account for the ablative material. It does this by calculating the average heating rate to node 15 over the time step just performed and utilizes it as an inner surface boundary condition for the internally constructed 1-D network representation of the ablative material. The correctness of this analytical approach can be rigorously substantiated for use with explicit network solution subroutines. However, when used with large time step implicit methods it yields a controlled instability and the results may be questionable. It is up to the user to determine the solution accuracy by whatever means available. A more complicated Variables 2 operations block could be as follows:

```
(Col) 1    5  8    12

            BCD 3VARIABLES 2
                DIDEG1(TIMEN,A10,K8)        $GET TEST TEMPERATURE
                SUB(T8,K8,TTEST)            $ØBTAIN TEMP DIFFERENCE
      F         IF(TTEST.LE.2.0) GØ TØ 10
                MLTPLY(G7,0.99,G7)          $REDUCE CØNDUCTANCE
          5     STFSEP(-1.0,BACKUP)         $SET BACKUP NØN-ZERØ
      F         GØ TØ 20
      F   10 IF(TTEST.GE.-2.0) GO TO 15
                MLTPLY(G7,1.01,G7)          $INCREASE ØONDUCTANCE
      F         GØ TØ 5
         15     QMETER(T8,T15,K9)
                QINTEG(K9,DTIMEU,K10)
      F   20 CØNTINUE
             END
```

This corresponds to a portion of a network as follows:



Array 10 is a time-temperature test history of node 8 and node 15 is a known boundary reference temperature. The problem is to calculate the value of conductor seven which will yield a calculated temperature at node eight that is within ±2.0 degrees of the test history. The above Variables 2 operations will attempt to modify conductor seven so that it will meet the constraints on temperature eight. It is quite "brute-force" and unsophisticated. However, the corrector operations are at the discretion of the user. If the tolerances were too severe or the correction operations too strong the correction for one tolerance could lead to dissatisfaction of the other and an impasse result. If the reference temperature at node 15 were incorrect, possibly no value of conductor seven would satisfy the constraints. The end result of such a study would be to produce a plot of conductance seven versus time which could be used to derive an empirical relationship with other parameters. Too wide a tolerance would cause the plot to resemble a stair-case function. Please note that either condition being unsatisfied causes control constant BACKUP to be set non-zero and the iteration to be redone with the corrected conductor seven value. Only when all criteria are met are the metering and integration operations performed.

4.2.7.5 Output Calls Operations Block

This operations block could have been entitled Variables 3 but Output Calls seemed more appropriate. In it a user may call upon any desired subroutine. However, its contents are performed on the output interval so it is only logical that it would primarily contain instructions for outputing information. There is a variety of output subroutines offering the user several format options. A very simple Output Calls block would be as follows:

```
(Col)   8    12

        BCD 3ØUTPUT CALLS
            PRNTMP
        END
```

The above call will output certain time control informtion and the temperature of every node in the network under consideration. The node temperatures will correspond to the relative node numbers set up by the preprocessor, not the actual node numbers set by the user. The preprocessor lists out all of the input data. Immediately after the input node data a dictionary of relative node numbers versus actual node numbers is listed. By utilizing it a user can correlate the relative node temperatures with his actual numbers.

In addition to the various subroutines for printing output, there are several plotting subroutines available. However, the plotting subroutines require that the information to be plotted exist as arrays. In order to plot transient temperatures versus time it is necessary for the user to store the information until the transient is completed and then perform plotting. The operations to do this could be as follows:

```
(Col)   8   12
        BCD 3ØUTPUT CALLS
            PRNTMP
            ADDFIX(1,K10,K10)
            STØARY(K10,A1,TIMEM)
            STØARY(K10,A2,T1)
        END
```

The Output Calls will be performed at problem start time and on the output interval until problem stop time is reached. A 100 minute transient with an output interval of 5 minutes would cause the Output Calls operations to be performed 21 times. With constant ten initially at zero, the ADDFIX call will add an integer one to it each time it is performed. The STØARY call causes the third arguments (TIMEM and T1) to be stored into the K10th location of array one and two respectively. Therefore, A1 and A2 must contain at least as many data locations as required to accommodate the STØARY operations. When the transient analysis is completed, A1 and A2 contain array data suitable for plotting or printing in a columnar format. Such operations are easily called for in the Execution Operations Block immediately following the network solution call.

The above data and operations blocks constitute a problem data deck which must be terminated by the following card:

```
(Col)   8   12

        BCD 3END ØF DATA
```

4.2.8 Parameter Runs

Parametric analysis which does not involve network or operations changes to the original problem may be performed on the same computer run. Only data values such as output page heading, temperatures, capacitances, conductance, constants and arrays may be changed. The data change blocks must all be specified whether changes occur in the block or not and the data input is identical to the preceding discussion with the exception of conductors. When specifying new conductances the adjoining node information is deleted; only the conductor number and value are required.

Two parametric run options are available, INITIAL and/or FINAL, and they may be used several times within the problem data deck. The problem data deck as initially input is referred to as the original problem. Any and all INITIAL parameter runs refer to it exactly as it was input. The FINAL parameter run refers to the just completed problem exactly as terminated. When two INITIAL parameter runs are attached to the end of a problem data deck, they both refer to the original problem at start time. However, when two FINAL parameter runs are attached to the end of a problem data deck, the first refers to the original as terminated, and the

second refers to the first FINAL parameter run as completed. The SINDA control cards necessary to specify a parameter run is as follows:

```
(Col)  8    12
       BCD 3INITIAL PARAMETERS
    or BCD 3FINAL PARAMETERS
       END
       BCD 3NØDE DATA
       END
       BCD 3CØNDUCTØR DATA
       END
       BCD 3CØNSTANTS DATA
       END
       BCD 3ARRAY DATA
       END
```

The parameter run decks are inserted in the problem data deck immediately preceding the BCD 3END ØF DATA card. After the BCD parameter card, the user may insert additional BCD data to replace the original problem output page heading. When changing an array, the entire new array must be input and be exactly the length of its original. Parameter runs conserve machine time mainly due to not having to reform the pseudo-compute sequence. If a user desires, he may accomplish FINAL parameter runs by calling the network execution subroutine twice in the EXECUTIØN block and inserting the necessary calls to modify data values between them.

4.2.9 Store and Recall Problem Options

The capability to store complete problems on and recall them from magnetic tape is a useful feature of SINDA. While the parameter run capability is useful for performing parametric analysis in the same run, the store and recall capability allows an indefinite time lapse between parametric analysis. In addition, long duration problems may be broken into several short duration runs. If a parametric analysis is such that the first portion of the runs are identical, then the problem can be run for the constant portion, stored and then recalled as many times as necessary.

The store problem feature is achieved by a user initiated subroutine call which is as follows:

```
(Col)      12
           STØREP(KX)
```

where KX refers to a constant location containing an alphanumeric identification name for the stored problem. The call may be used as many times as desired but the user must insure that each activation references a unique name. It is up to the user to insure that the stored problem tapes have been mounted with the "write" ring in, are properly positioned and that the computer operator has been instructed to save the tapes. The user should check Appendix E, Control Cards and Deck Setup to determine which tapes his problem

is being stored on and the control cards, if any, for assigning it within the system.

The recall problem feature is a SINDA preprocessor option which is activated by the following card;

(Col) 1 13

RECALL AAAAAA

where AAAAAA is the alphanumeric identification name of the stored problem. This single card replaces the blank card preceding the problem data deck and must be followed by initial parameter and block data change cards exactly as shown for parameter runs, including the first BCD 3 parameter and END cards and also the BCD 3END ØF DATA card. The stored problem identified will be searched for and brought into core from the two storage tapes. Any data changes specified will be performed and then control is passed to the first subroutine call in the EXECUTIØN block. The user must remember that the recalled problem contains the STØREP call. The user is again advised to consult Section E for the tape unit designations, control card requirements and operator instructions necessary for mounting the stored problem tape.

4.2.10 Dictionary Printout

4.2.10.1 Node Data

A dictionary of relative vs. actual node numbers is automatically printed under BCD 3NØDE DATA BLØCK.

4.2.10.2 Conductor Data

A dictionary of relative vs. actual conductor numbers is automatically printed under BCD 3CØNDUCTØR DATA BLØCK.

4.2.10.3 Constants Data

A dictionary of relative vs. actual constants numbers printout is optional under BCD 3CØNSTANTS DATA BLØCK. For a printout, a * in Col. 80 is used.

(Col) 8 12 80

BCD 3CØNSTANTS DATA *

4.2.10.4 Array Data

A dictionary of actual array number vs. FØRTRAN addresses printout is optional under BCD 3ARRAY DATA.

(Col) 8 12 80

BCD 3ARRAY DATA *

5. ERROR MESSAGES

Due to the variety of subroutines available and the variable number of arguments which some of them have, no check is made to determine if a subroutine has the correct number of arguments. An incorrect number of arguments on a subroutine call will generally cause job termination immediately after successful compilation, usually without any error message. If the above occurs, the user should closely check the number of arguments for his subroutine calls.

Numerous error messages can be output by the preprocessor. These error messages are listed below and grouped according to various preprocessor functions. All error messages are preceded by three asterisks which have been deleted below. Self-explanatory messages are not enlarged upon; note that all messages are outputted on one line except 5.1.24.

5.1 Processing Data Blocks

5.1.1 DATA BLØCKS IN IMPRØPER ØRDER ØR ILLEGAL BLØCK DESIGNATIØN ENCØUNTERED

5.1.2 THE PSEUDØ CØMPUTE SEQUENCE INDICATØR MUST BE EITHER SPCS ØR LPCS, AND START IN CØLUMN 21

5.1.3 AN IMBEDDED BLANK HAS BEEN ENCØUNTERED IN THE LAST LINE

5.1.4 BLANK CØUNT ØF TEN HAS BEEN EXCEEDED

5.1.5 INTEGER FIELD EXCEEDS 10

5.1.6 REAL NUMBER FIELD EXCEEDS 20

5.1.7 ALPHANUMERIC FIELD EXCEEDS 6

5.1.8 CØND NUMBER, XXXXX, IS THE DUPLICATE ØF THE XXXXXTH CØNDUCTØR

5.1.9 MULTIPLE DECIMAL PØINTS HAVE BEEN ENCØUNTERED

5.1.10 TWØ CØNSECUTIVE CØNDUCTØR VALUES HAVE BEEN ENCØUNTERED

5.1.11 THE NØDE NUMBER, ENTRY XXX, MUST BE AN INTEGER

5.1.12 THE TEMPERATURE VALUE, ENTRY XXX, MUST BE A FLØATING PØINT NUMBER

5.1.13 THE CAPACITANCE VALUE, ENTRY XXX, MUST BE A FLØATING PØINT NUMBER

5.1.14 THE NUMBER OF NØDES, ENTRY XXX, MUST BE A PØSITIVE INTEGER

5.1.15 THE NØDE INCREMENT, ENTRY XXX, MUST BE A NØN-ZERØ INTEGER

5.1.16 THE ABØVE CARD HAS XXX ENTRIES, THE NUMBER ØF ENTRIES MUST BE A MULTIPLE ØF XXX

5.1.17 THE ABØVE CARD HAS XXX ENTRIES, THE NUMBER ØF ENTRIES MUST BE A MULTIPLE OF XXX ØR XXX

5.1.18 NØDE NUMBER XXXXX HAS BEEN DEFINED TWICE AT RELATIVE LØCATIØNS XXXXX AND XXXXX

5.1.19 CØLUMNS 8,9,10 CØNTAIN THE ILLEGAL CØDE XXX

5.1.20 THE ARRAY SPECIFICATIØN, ENTRY XXX, MUST BEGIN WITH THE LETTER A

5.1.21 THE CØNSTANT SPECIFICATIØN, ENTRY XXX, MUST BE EITHER A FLØATING PØINT NUMBER ØR BEGIN WITH THE LETTER K

5.1.22 THE ARRAY ØR CØNSTANT IDENTIFICATIØN XXXX MUST BE A PØSITIVE INTEGER

5.1.23 THE NØDE NUMBER MUST BE GREATER THAN 0 FØR THIS ØPTIØN

5.1.24 BØTH ARRAY SPECIFICATIØNS, ENTRIES XXX AND XXX, ARE FLØATING PØINT NUMBERS

AT LEAST ØNE OF THESE MUST IDENTIFY AN ARRAY NUMBER

5.1.25 THE CØNDUCTØR NUMBER, ENTRY XXX, MUST BE AN INTEGER

5.1.26 THE CØNDUCTANCE VALUE, ENTRY XXX, MUST BE A FLØATING PØINT NUMBER

5.1.27 ACTUAL NØDE NUMBER XXXXX WAS NØT SPECIFIED IN THE NØDE DATA BLØCK

5.1.28 THE NUMBER ØF CØNDUCTØRS, ENTRY XXX, MUST BE A PØSITIVE INTEGER

5.1.29 THE CØNDUCTØR INCREMENT, ENTRY XXX, MUST BE A NØN-ZERØ INTEGER

5.1.30 THE NØDE INCREMENT, ENTRY XXX, MUST BE INTEGER

5.1.31 ENTRY XXX IS ASSUMED TØ BE A FIXED CØNSTANT NAME, BUT THE NAME INPUT IS NØT IN THE LIST ØF FIXED CØNSTANTS

5.1.32 CØNSTANT NUMBER XXXXX IS THE DUPLICATE ØF THE XXXXX RELATIVE CØNSTANT

5.1.33 ARRAY NUMBER XXXXX HAS ALREADY BEEN INPUT AS RELATIVE NUMBER XXXXX

5.1.34 TEMPERATURE VARYING CAPACITANCE ENTRY XXXXX IN NØDE DATA SPECIFIES ARRAY XXXXX WHICH IS NØT IN THE LIST

5.1.35 TEMPERATURE VARYING CAPACITANCE ENTRY XXXXX IN NØDE DATA SPECIFIES CØNSTANT XXXXX WHICH IS NØT IN THE LIST

5.1.36 TEMPERATURE VARYING CØNDUCTANCE ENTRY XXXXX IN CØNDUCTØR DATA SPECIFIES ARRAY XXXXX WHICH IS NØT IN THE LIST

5.1.37 TEMPERATURE VARYING CØNDUCTANCE ENTRY XXXXX IN CØNDUCTØR DATA SPECIFIES CØNSTANT XXXXX WHICH IS NØT IN THE LIST

5.1.38 THE PRØGRAM EXPECTED ENTRY XXX ABØVE TØ BE AN INTEGER ARRAY NUMBER. A NØN-INTEGER WAS ENCØUNTERED.

5.2 Forming Pseudo Compute Sequence

5.2.1 RELATIVE NØDE NUMBER (XXXXX) IS NØT CØNNECTED TØ ANY ØTHER NØDE

5.3 Processing Program Blocks

5.3.1 EXECUTIØN BLØCKS IN IMPRØPER ØRDER ØR ILLEGAL BLØCK DESIGNATIØN ENCØUNTERED

5.3.2 VARIABLE DESIGNATØR, AAA, NØT DEFINED FØR GENERAL PRØBLEM

Explanation: Some alpha character other than K or A has been used to reference a data block. In a thermal problem a designator other than G, K, or A is assumed to be referencing the nodal block.

5.3.3 MISSING NØDE NUMBER, XXXXX

5.3.4 MISSING CØNDUCTØR NUMBER, XXXXX

5.3.5 MISSING CØNSTANT NUMBER, XXXXX

5.3.6 MISSING ARRAY NUMBER, XXXXX

5.3.7 FIXED CONSTANT NAME, AAAAA, NØT IN LIST.

5.3.8 NUMBER ØF SUBRØUTINES REQUESTED EXCEEDS 75.

Explanation: More than 75 unique subroutines have been called.

5.4 Processing Parameter Changes

5.4.1 NØDE NUMBER, XXXXX, WAS NØT DEFINED IN THE ØRIGINAL PRØBLEM.

5.4.2 CØNDUCTØR NUMBER, XXXXX, WAS NØT DEFINED IN THE ØRIGINAL PRØBLEM.

5.4.3 CØNSTANT NUMBER, XXXXX, WAS NØT DEFINED IN THE ØRIGINAL PRØBLEM.

5.4.4 ARRAY NUMBER, XXXXX, WAS NØT DEFINED IN THE ØRIGINAL PRØBLEM.

5.4.5 CØNSTANTS BLØCK WAS EMPTY IN THE ØRIGINAL PRØBLEM.

5.4.6 ARRAY BLØCK WAS EMPTY IN THE ØRIGINAL PRØBLEM.

5.4.7 THE ABØVE ARRAY IS LØNGER THAN THE ARRAY DEFINED IN THE ØRIGINAL PRØBLEM

5.4.8 NØDE ØR CØNDUCTØR DATA IS NØT ALLØWED IN A GENERAL PRØBLEM

5.5 Terminations Due to Errors (No Preceding Asterisks)

5.5.1 ERRØR TERMINATIØN - LØADING IS SUPPRESSED

5.5.2 THE NUMBER ØF ERRØR MESSAGES EXCEEDS 200 - RUN TERMINATED

6. REFERENCES

1. Gaski, J. D. and Lewis, D. R., "Chrysler Improved Numerical Differencing Analyzer", TN-AP-66-15, April 30, 1966, Chrysler Corporation Space Division, New Orleans, Louisiana.

2. Lewis, D. R., Gaski, J. D. and Thompson, L. R., "Chrysler Improved Numerical Differencing Analyzer for 3rd Generation Computers", TN-AP-67-287, October 20, 1967, Chrysler Corporation Space Division, New Orleans, Louisiana.

3. Karplus, W. J., Analog Simulation, Solution of Field Problems, McGraw-Hill Book Co., 1958.

4. Ishimoto, T. and Bevans, J. T., "Method of Evaluating Script F for Radiant Exchange within an Enclosure", AIAA Journal, Vol. 1, No. 6, June 1963, pp. 1428-1429.

5. Bobco, R. P., "Radiation Heat Transfer in Semigray Enclosures with Specularly and Diffusely Reflecting Surfaces", Journal of Heat Transfer, Vol. 86, 1964, pp. 123-130.

6. Clippinger, R. F. and Levin, J. H., "Numerical Analysis", Handbook of Automation, Computation, and Control, Vol. 1, Edited by Grabbe, E. M., Ramo, S., Wooldridge, D.E., pg. 14-84, 1958, John Wiley & Sons.

7. Crank, J. and Nicholson, P., "A Practical Method for Numerical Integration of Solutions of Partial Differential Equations of Heat-Conduction Type, Proc. Cambridge Phil. Soc. 43,50 (1947).

A. SINDA SUBROUTINES

A.1 Alphabetical Listing

*Internal, STEP Subroutine; **Internal, SC-4060 Plot Pkg.;***Internal

A.2 EXECUTION SUBROUTINES (NETWORK SOLUTION & OUTPUT)

NETWORK SOLUTION -- STEADY STATE

EXECUTION SUBROUTINE NAME: CINDSS*

PURPOSE:

This subroutine ignores the capacitance values of diffusion nodes to calculate the network steady state solution. Due to the SPCS requirement, diffusion nodes are solved by a "block" iterative method. However, if all diffusion nodes were specified as arithmetic nodes they would be calculated by a "successive point" iterative method. The user is required to specify the maximum number of iterations to be performed in attempting to reach the steady state solution (control constant NLØØP) and the relaxation criteria which determines when it has been reached (DRLXCA for diffusion nodes and/or ARLXCA for arithmetic nodes). The subroutine will continue to iterate until one of the above criteria is met. If the iteration count exceeds NLØØP an appropriate message is printed. Variables 1 and Output Calls are performed at the start and Variables 2 and Output Calls are performed upon completion. If not specified, control constants DAMPD and DAMPA are set at 1.0. They are used as multipliers times the new temperatures while 1.0 minus their value is used as multipliers times the old temperatures in order to "weight" the returned answer. This weighting of so much new and so much old is useful for damping oscillations due to nonlinearities. They may also be used to achieve over relaxation.

If a series of steady state solutions at various times are desired it can be accomplished by specifying control constants TIMEND and ØUTPUT. ØUTPUT will be used both as the output interval and the computation interval. In this case appropriate calls would have to be made in Variables 1 to modify boundary conditions with time.

If desired, the CINDSS call can be followed by a call to one of the transient solution subroutines which has the same SPCS requirement. In this manner the steady state solution becomes the initial conditions for the transient analysis. However, since CINDSS utilizes control constants TIMEND and ØUTPUT the user must specify their values in the execution block after the steady state call and prior to the transient analysis call.

RESTRICTIONS:

The SPCS option is required. Diffusion nodes receive a "block" iteration while arithmetic nodes receive a "successive point" iteration, no acceleration features are utilized. Control constants NLØØP and DRLXCA and/or ARLXCA must be specified. Successive steady state solutions can be obtained by specifying control constants TIMEND and ØUTPUT. Other control constants which are activated or used are: LØØPCT, DRLXCC and/or ARLXCC, TIMEN, TIMEM, TIMEØ, DAMPD, DAMPA, DTIMEU, LINECT and PAGECT. Control constant ØPEITR is checked for output each iteration.

CALLING SEQUENCE: CINDSS

*This subroutine utilizes one dynamic storage core location for each diffusion node.

NETWORK SOLUTION -- STEADY STATE

EXECUTION SUBROUTINE NAME: CINDSL*

PURPOSE:

This subroutine ignores the capacitance values of diffusion nodes to calculate the network steady state solution. Since this subroutine has the LPCS requirement, both diffusion and arithmetic nodes receive a "successive point" iteration. In addition, on every third iteration, a linear extrapolation is performed on the error function plot of each node in an attempt to accelerate convergence. The user is required to specify the maximum number of iterations to be performed in attempting to reach the steady state solution (control constant NLØØP) and the relaxation criteria which determines when it has been reached (DRLXCA for diffusion nodes and/or ARLXCA for arithmetic nodes). The subroutine will continue to iterate until one of the above criteria is met. If the iteration count exceeds NLØØP an appropriate message is printed. Variables 1 and Output Calls are performed at the start and Variables 2 and Output Calls are performed upon completion. If not specified, control constants DAMPD and DAMPA are set at 1.0. They are used as multipliers times the new temperatures while 1.0 minus their value is used as multipliers times the old temperatures in order to "weight" the returned answer. This weighting of so much new and so much old is useful for damping oscillations due to nonlinearities. They may also be used to achieve over relaxation.

If a series of steady state solutions at various times are desired it can be accomplished by specifying control constants TIMEND and ØUTPUT. ØUTPUT will be used both as the output interval and the computation interval. In this case appropriate calls would have to be made in Variables 1 to modify boundary conditions with time.

If desired, the CINDSL call can be followed by a call to one of the transient solution subroutines which has the same LPCS requirement. In this manner the steady state solution becomes the initial conditions for the transient analysis. However, since CINDSL utilizes control constants TIMEND and ØUTPUT the user must specify their values in the execution block after the steady state call and prior to the transient analysis call.

RESTRICTIONS:

The LPCS option is required. Diffusion and arithmetic nodes receive a "successive point" iteration and an extrapolation method of acceleration. Control constants NLØØP and DRLXCA and/or ARLXCA must be specified. Successive steady state solutions can be obtained by specifying control constants TIMEND and ØUTPUT. Other control constants which are activated or used are: LØØPCT, DRLXCC, and/or ARLXCC, TIMEN, TIMEM, TIMEØ, DAMPD, DAMPA, DTIMEU, LINECT and PAGECT. Control constant ØPEITR is checked for output each iteration.

CALLING SEQUENCE: CINDSL

*This subroutine utilizes two dynamic storage core locations for each diffusion and arithmetic code.

NETWORK SOLUTION -- STEADY STATE

EXECUTION SUBROUTINE NAME: CINDSM

PURPOSE:

This is a steady state execution subroutine specifically designed for radiation dominated problems. The CINDSL subroutine is the base and was modified to operate in a quasi-linear manner. The problem is linearized (i.e., effective radiation evaluated and held constant) and then the linearized problem is solved. The nonlinearities are then revaluated and fixed (linearized) and the problem is again solved. This linearization frequency is based on a new control constant LAXFAC (an integer). The user must satisfy the control constant requirements for CINDSL.

RESTRICTIONS:

The long psuedo-compute sequence is required, control constant LAXFAC must be specified. See subroutine CINDSL.

CALLING SEQUENCE: CINDSM

NETWORK SOLUTION -- TRANSIENT

EXECUTION SUBROUTINE NAMES: CNFRWD* or CNFRDL*

PURPOSE:

These subroutines perform transient thermal analysis by the explicit forward differencing method. The stability criteria of each diffusion node is

$$C_i * \left[T_i(new) - T_i(old) \right] = \left\{ \sum_{\substack{i=0 \\ j \neq i}}^{n} G_j * \left[T_j(old) - T_i(old) \right] + Q_i \right\} \Delta t$$

G_j represents the conductors into node i, Q_i the source location, and C_i the nodal capacity

The stability criteria of each diffusion node is calculated and the minimum value is placed in control constant CSGMIN. The time step used (control constant DTIMEU) is calculated as 95% of CSGMIN divided by CSGFAC. Control constant CSGFAC is set at 1.0 unless specified larger by the user. A "look ahead" feature is used when calculating DTIMEU. If one time step will pass the output time point the time step is set to come out exactly on the output time point; if two time steps will pass the output time point, the time step is set so that two time steps will come out exactly on the output time point. DTIMEU is also compared to DTIMEH and DTIMEL. If DTIMEU exceeds DTIMEH it is set equal to it; if DTIMEU is less than DTIMEL the problem is terminated. If no input values are specified, DTIMEL is set at zero and DTIMEH it is set at infinity. The maximum temperature change calculated over an iteration is placed in control constant DTMPCC and/or ATMPCC. They are compared to DTMPCA and/or ATMPCA respectively and if larger cause DTIMEU to be modified so that they campare as equal to or less than DTMPCA and/or ATMPCA. If DTMPCA and/or ATMPCA are not specified they are set at infinity.

All diffusion nodes are calculated prior to solving the arithmetic nodes. The user may iterate the arithmetic node solution by specifying control constants NLØØP and ARLXCA. If the arithmetic node iteration count exceeds NLØØP the answers are accepted as is, and the subroutine continues without any user notification. In addition, the user may specify control constant DAMPA in order to dampen possible oscillations due to nonlinearities. The arithmetic nodes may be used to specify an incompressible pressure or radiosity network. In this manner they would be solved implicity each time step but evaluation of temperature varying properties would suffer a one time step lag.

RESTRICTIONS:

The SPCS option is required for CNFRWD, the LPCS option is required for CNFRDL, and control constants TIMEND and ØUTPUT must be specified. Problem start time, if other than zero, may be specified as TIMEØ. Other control constants used or activated are: TIMEN, TIMEM, CSGMIN, CSGFAC, DTIMEU, DTIMEH, DTMPCA, DTMPCC, ATMPCA, ATMPCC, NLØØP, LØØPCT, DAMPA, ARLXCA, ARLXCC, ØPEITR, BACKUP, LINECT and PAGECT.

CALLING SEQUENCE: CNFRWD or CNFRDL

* These subroutines utilize one dynamic storage core location for each diffusion and arithmetic node.

NETWORK SOLUTION -- TRANSIENT

EXECUTION SUBROUTINE NAME: CNFAST*

PURPOSE:

This subroutine is a modified version of CNFRWD which allows the user to specify the minimum time step to be taken. The time step calculations proceed exactly as in CNFRWD until the check with DTIMEL is made. If DTIMEU is less than DTIMEL it is set equal to it. As each node is calculated its CSGMIN is obtained and compared to DTIMEU. If equal to or greater, the nodal calculation is identical to CNFRWD. If the CSGMIN for a node is less than DTIMEU the node receives a steady state calculation. If only a small portion of the nodes in a system receive the steady state calculation the answers are generally reasonable. However, as the number of nodes receiveing steady state calculations increases, so do the solution inaccuracies.

RESTRICTIONS:

The SPCS option is required and control constants TIMEND and ØUTPUT must be specified. The checks on control constants DTMPCA, ATMPCA and BACKUP are not performed. Other control constants which are used or activated are: TIMEN, TIMEH, TIMEØ, CSGMIN, CSGFAC, DTIMEU, DTIMEL, DTIMEH, DTMPCC, ATMPCC, DAMPA, ARLXCA, ARLXCC, NLØØP, LØØPCT, LINECT and PAGECT.

CALLING SEQUENCE: CNFAST

* This subroutine utilizes one dynamic storage core location for each diffusion node.

NETWORK SOLUTION -- TRANSIENT

EXECUTION SUBROUTINE NAME: CNEXPN*

PURPOSE:

This subroutine performs transient thermal analysis by the exponential prediction method and the solution equation is of the following form:

$$T_i' = \left(\frac{\Sigma G_j T_j \quad Q_i}{\Sigma G_j} \right) \left(1 - e^{-\frac{\Sigma G_j \Delta t}{C_i}} \right) + T_i e^{-\frac{\Sigma G_j \Delta t}{C_i}}$$

For the derivation the reader should note the reference below. The above equation is unconditionally stable no matter what size time step is taken and reduces to the steady state equation for an infinite time step. However, stability is not to be confused with accuracy. Time steps larger than those taken with CNFRWD remain stable but tend to lose or gain energy in the system. For this reason, this subroutine is not recommended where accuracy is sought. However, it is suitable for parametric analysis where trends are sought and a more accurate method will be utilized for a final analysis.

The inner workings of the subroutine are virtually identical to CNFRWD with the exception of the solution equation and the use of CSGFAC. The time step used (DTIMEU) is calculated as CSGMIN times CSGFAC. The lock ahead feature for calculating the time step is identical as are the checks with DTIMEH, DTIMEL and DTMPCA. The diffusion nodes are calculated prior to the arithmetic nodes and the arithmetic nodes utilize NLØØP, ARLXCA and DAMPA exactly the same as CNFRWD.

RESTRICTIONS:

The SPCS option is required and control constants TIMEND and ØUTPUT must be specified. Problem start time if other than zero may be specified as TIMEØ. Other control constants used or activated are: TIMEN, TIMEM, CSGMIN, CSGFAC, DTIMEU, DTIMEL, DTIMEH, DTMPCA, DTMPCC, ATMPCA, ATMPCC, ARLXCA, ARLXCC, DAMPA, ØPEITR, BACKUP, LINECT and PAGECT.

CALLING SEQUENCE: CNEXPN

*This subroutine utilizes one dynamic storage core location for each diffusion and arithmetic node.

Ref: Gaski, J. D. and Lewis, D. R., "Chrysler Improved Numerical Differencing Analyzer," TN-AP-66-15, April 30, 1965, Page 5.1.3.

NETWORK SOLUTION — TRANSIENT

EXECUTION SUBROUTINE NAME: CNDUFR

PURPOSE:

This subroutine performs an unconditionally stable explicit finite differencing solution often called the Du Fort-Frankel method. This is basically the forward differencing equation but the present temperature of the node being operated on is replaced by a time weighted average of future and past temperatures. This substitution is performed on the space derivative temperatures only. The user may specify time steps larger than stability criteria, but within reason.

RESTRICTIONS:

The same as CNEXPN, CSGFAC is used as a factor (>1.0) to increase the time step used above the stability limit.

CALLING SEQUENCE: CNDUFR

Ref: DuFort, E. C. and Frankel, S. P., "Stability Conditions in the Numerical Treatment of Parabolic Differential Equations," Mathematical Tables and Other Aids to Computation, Vol. 7-8, 1953-54, pp 135-152.

NETWORK SOLUTION -- TRANSIENT

EXECUTION SUBROUTINE NAME: CNFWBK*

PURPOSE:

This execution subroutine performs transient thermal analysis by implicit "forward-backward" finite differencing (Crank-Nicholson Method).

$$2C_i*\left[T_i(\text{new})-T_i(\text{old})\right] = \left\{ \sum_{\substack{j=0 \\ j\neq i}}^{n} G_j*\left[T_j(\text{new})+T_j(\text{old})-T_i(\text{new})-T_i(\text{old})\right]+2Q_i \right\}*\Delta t$$

The LPCS option is required and allows the simultaneous set of equations to be solved by "successive point" iterations. During the first iteration for a time step, the capacitance values are doubled and divided by the time step and the energy transfer rates based on old temperatures are added to the source locations. Upon completing the time step the capacitance values are returned to their original state. The iteration looping, convergence criteria and other control constant checks are identical to CNBACK. The time step checks and calculations and look ahead feature are identical to that used for CNBACK.

The automatic radiation transfer damping and extrapolation method of acceleration mentioned under the CNBACK subroutine writeup are also employed in this subroutine. Diffusion and/or arithmetic temperature calculations may be damped through use of DAMPD and/or DAMPA respectively. Control constants BACKUP and ØPEITR are continuously checked. CNFWBK internally performs forward-backward differencing of boundary conditions. For this reason the user should utilize TIMEN as the appropriate independent variable in Variables 1 operations.

It is interesting to note the CNFWBK generally converges in 25% fewer iterations than CNBACK. The probable reason for this is that the boundary of the mathematical system is better defined. While every future temperature node under CNBACK is connected to its present temperature, under CNFWBK every future temperature node is also receiving an impressed source based on the present temperature.

RESTRICTIONS:

The LPCS option is required. Control constants TIMEND, ØUTPUT DTIMEI NLØØP and DRLXCA and/or ARLXCA must be specified. Other control constants which are used or activated are: TIMEN, TIMEØ, TIMEM, CSGMIN, DTIMEU, DTIMEH, DTMPCA, DTMPCC, ATMPCA, ATMPCC, DAMPD, DAMPA, DRLXCC and/or ARLXCC, LØØPCT, BACKUP, ØPEITR, LINECT and PAGECT.

CALLING SEQUENCE: CNFWBK

*This subroutine utilizes three dynamic storage core locations for each diffusion node and one for each arithmetic and boundary node.

NETWORK SOLUTION -- TRANSIENT

EXECUTION SUBROUTINE NAME: CNBACK*

PURPOSE:

This subroutine performs transient thermal analysis by implicit backward differencing.

$$C_i * \left[T_i(new) - T_i(old)\right] = \left\{ \sum_{\substack{j=0 \\ j \neq i}}^{n} G_j * \left[T_j(new) - T_i(new)\right] + Q_i \right\} * \Delta t$$

The LPCS option is required and allows the simultaneous set of equations to be solved by "successive point" iteration. Each third iteration, diffusion node temperatures which trace a continuous decreasing slope receive an extrapolation on their error function curve in an attempt to accelerate convergence. For convergence criteria the user is required to specify NLØØP and DRLXCA and or ARLXCA. If the number of iterations during a time step exceeds NLØØP a message is printed but the problem proceeds.

Variables 1 is performed only once for each time step. Since this subroutine is implicit the user must specify the time step to be used as DTIMEI in addition to TIMEND and ØUTPUT. The look ahead feature for the time step calculation in CNFRWD is used as are the checks for DTIMEH, DTMPCA and ATMPCA but not DTIMEL. Damping of the solutions can be achieved through use of control constants DAMPD and/or DAMPA. Control constants BACKUP and ØPEITR are continuously checked.

Implicit methods of solution often oscillate at start up or for boundary step changes when radiation conductors are present. CNBACK contains an automatic damping feature which is applied to radiation conductors. The radiation transfer to a node is calculated for its present temperature and a temporary new temperature is calculated. Then the radiation transfer is recalculated and the final node temperature is calculated based on the arithmetic mean of the two radiation transfer calculations. This automatic radiation damping has proven to be quite successful and lessens the need for use of DAMPD and DAMPA.

RESTRICTIONS:

The LPCS option is required. Control constants TIMEND, ØUTPUT, DTIMEI, NLØØP and DRLXCA and/or ARLXCA must be specified. Other control constants which are used or activated are: TIMEN, TIMEØ, TIMEM, CSGMIN, DTIMEV, DTIMEH, DTMPCA, DTMPCC, ATMPCA, ATMPCC, DAMPD, DAMPA, DRLXCC and/or ARLXCC, LØØPCT, BACUP, ØPEITR, LINECT and PAGECT.

CALLING SEQUENCE: CNBACK

*This subroutine utilizes three dynamic storage core locations for each diffusion node and one for each arithmetic and boundary node.

NETWORK SOLUTION -- TRANSIENT

EXECUTION SUBROUTINE NAME: CNVARB

PURPOSE:

This subroutine applies an implicit finite differencing solution to the diffusion equation. It internally calculates a variable beta weighting factor (see equation 3-3, page 3-3) as the ratio of the explicit stability criteria, CSGMIN, divided by the computation time step used, DTIMEU. A constraint that beta must be equal to or larger than one half is imposed. Hence, the method of solution lies somewhere between backward and forward-backward differencing.

RESTRICTIONS:

The restrictions listed for CNFWBK and/or CNBACK apply.

CALLING SEQUENCE: CNVARB

NETWORK SOLUTION - - TRANSIENT

EXECUTION SUBROUTINE NAME: CNQUIK

PURPOSE:

This is an unconditionally stable explicit method of solution which allows SINDA users to employ computation intervals larger than CNFRWD. The method of solution is a 50-50 combination of exponential predictions (CNEXPN) and DuFort-Frankel (CNDUFR). For a temperature rising situation the CNEXPN routine tends to undershoot while CNDUFR tends to overshoot; however, CNQUIK falls between the two and generally yields better results than either CNEXPN or CNDUFR.

RESTRICTIONS:

The short pseudo-compute sequence is required. The control constant requirements for CNEXPN or CNDUFR apply to CNQUIK.

CALLING SEQUENCE: CNQUIK

OUTPUT

EXECUTION SUBROUTINE NAMES: CSGDMP or RCDUMP

PURPOSE:

These subroutines are designed to aid in the checkout of thermal problem data decks. They call upon Variables 1 (CSGDMP also calls upon Output Calls) and then print out each actual diffusion node number with the capacitance and CSGMIN value of the node. For each node they identify, the attached conductors by actual conductor number, list the type and conductance value and the actual number and type of the adjoing node. Either the SPCS or LPCS option may be used. While the LPCS option allows every conductor attached to a node to be identified, the SPCS option only identifies conductors for the first node number on which they occur. After the diffusion nodes are processed the connection information for the arithmetic nodes is listed. After listing the above information control passes to the next sequentially listed subroutine or CSGDMP from Output Calls.

RESTRICTIONS:

The CSGDMP subroutine is called in the Execution block, while RCDUMP can be called from the Output Calls block. Never call either subroutine from Variables 1 or CSGDMP from Output Calls.

CALLING SEQUENCE: CSGDMP

or RCDUMP

A3. ARITHMETIC SUBROUTINES

Addition Operation

ADD	Sums a variable number of floating point numbers	A.3-4
ADDFIX	Sums a variable number of integer numbers	"
ADDARY	Adds the corresponding elements of two specified length arrays to form a third array	"
ARYADD	Adds a constant value to every element in an array to form a new array	"
SUMARY	Sums an array of floating point values	"

Subtraction Operation

SUB	Subtracts a variable number of floating point numbers	A.3-5
SUBFIX	Subtracts a variable number of integer numbers	"
SUBARY	Subtracts the corresponding elements of one array from another to form a third array	"
ARYSUB	Subtracts a constant value from every element in an array to form a new array	"

Multiplication Operation

MLTPLY	Multiplies a variable number of floating point numbers	A.3-6
MPYFIX	Multiplies a variable number of integer numbers	"
MPYARY	Multiplies the corresponding elements of two arrays to form a third.	"
ARYMPY	Multiplies each element of an array by a constant value to form a new array	"
SCLDEP, SCLIND	Multiplies the dependent or independent variables of a doublet type interpolation array	A.3-7
CMPXMP, CMPYI	Multiplies two complex numbers on the corresponding elements of arrays of complex numbers	"

Division Operation

DIVIDE	Performs a division of floating point numbers	A.3-8
DIVFIX	Performs a division of integer numbers	"
DIVARY	Divides the elements of one array into the corresponding elements of another array to produce a third array	"
ARYDIV	Divides each element of an array by a constant value to produce a new array	"
ARYINV	Inverts each element of an array in its own location	A.3-9
ARINDV	Divides each element of an array into a constant value to form a new array	"
ADDINV	Calculates one over the sum of the inverses of a variable number of arguments	"
ADARIN	Calculates one over the sum of inverses of an array of values	"
CMPXDV, CDIVI	Divides two complex numbers or the corresponding elements of arrays of complex numbers	A.3-10

Integer/Floating Point Conversion

FLØAT	Converts an integer to a floating point number	A.3-10
FIX	Converts a floating point number to an integer	"
INTRFC	Fractures a floating point number to yield the largest integer value possible and the remainder as a floating point number	"

Sign Conversion

SETPLS	Sets the sign positive for a variable number of arguments	A.3-11
ARYPLS	Sets the sign positive for data in a specified length array	"
SETMNS	Sets the sign negative for a variable number of arguments	"
ARYMNS	Sets the sign negative for every data value in a specified length array	"

Distribution of Array Data

SHFTV	Shifts a sequence of data from one array to another	A.3-12
SHFTVR	Shifts a sequence of data from one array and place data in reverse order in another array	"
FLIP	Reverses an array in its own array location	"
GENARY	Generates an array of equally incremented ascending values	"
BLDARY	Builds an array from a variable number of arguments in the order listed	A.3-13
BRKARY, BKARAD	Distributes values from within an array to a variable number of arguments in the order listed	"
STØARY	Places a value into or takes a value out of a specific array location	"
ARYSTØ		"
STFSEP	Places a constant value into a variable number of locations	A.3-14
SCALE	Utilizes a constant value to multiply a variable number of arguments	"
STFSEQ	Stuffs a constant value into a specified length array or group of sequential locations	"
STFSQS		"
SLDARY	Moves array data values back one or two positions and updates the last one or two values	A.3-15
SLDARD		"
STØRMA	Constructs historical data arrays during a transient analysis	"

Singlet/Doublet Array Generation

SPLIT	Separates a doublet array into two singlet arrays	A.3-16
JØIN	Combines two singlet arrays into a doublet array	"
SPREAD	Applies interpolation subroutine D1D1DA to two singlet arrays to obtain an array of dependent variables versus an array of independent variables	

Comparison Operation

MAXDAR MXDRAL	Obtains the absolute maximum difference between corresponding elements of two arrays of equal length N	A.3-17 "

ADDITION OPERATION

SUBROUTINE NAMES: ADD or ADDFIX

PURPOSE:

To sum a variable number of floating point or integer numbers respectively.

$$S = \Sigma\ Xi \quad , \quad i = 1,2,3,\ldots,N \quad , \quad N \geq 2$$

RESTRICTIONS:

Subroutine ADD is for floating point numbers while subroutine ADDFIX is for integers.

CALLING SEQUENCE: ADD(X1,X2,X3,...,XN,S)

or ADDFIX(X1,X2,X3,...,XN,S)

SUBROUTINE NAMES: ADDARY or ARYADD

PURPOSE:

Subroutine ADDARY will add the corresponding elements of two specified length arrays to form a third array. Subroutine ARYADD will add a constant value to every element in an array to form a new array. Their respective operations are:

$$Ai = Bi + Ci \quad , \quad i = 1,N$$

or

$$Ai = Bi + C \quad , \quad i = 1,N$$

RESTRICTIONS:

All data values to be operated on must be floating point numbers. The array length N must be an integer.

CALLING SEQUENCE: ADDARY(N,B(DV),C(DV),A(DV))

or ARYADD(N,B(DV),C,A(DV))

The answer array may be overlayed into one of the input array areas.

SUBROUTINE NAME: SUMARY

PURPOSE:

To sum an array of floating point values:

$$S = \Sigma\ Ai \quad , \quad i = 1,N$$

RESTRICTIONS:

The values to be summed must be floating point numbers and the array length N must be an integer.

CALLING SEQUENCE: SUMARY (N,A(DV),S)

SUBTRACTION OPERATION

SUBROUTINE NAMES: SUB or SUBFIX

PURPOSE:

To subtract a variable number of floating point or integer numbers respectively.

$$R = Y - \Sigma Xi \ , \ i = 1,2,3,\ldots,N \ , \ N \geq 1$$

RESTRICTIONS:

Subroutine SUB is for floating point numbers while the subroutine SUBFIX is for integers.

CALLING SEQUENCE: SUB(Y,X1,X2,X3,...,XN,R)

or SUBFIX(Y,X1,X2,X3,...,XN,R)

SUBROUTINE NAMES: SUBARY or ARYSUB

PURPOSE:

Subroutine SUBARY will subtract the corresponding elements of one array from another to form a third array. Subroutine ARYSUB will subtract a constant value from every element in an array to form a new array. Their respective operations are:

$$Ai = Bi - Ci \quad , \quad i = 1,N$$

or $$Ai = Bi - C \quad , \quad i = 1,N$$

RESTRICTIONS:

All data values to be operated on must be floating point numbers. The array length N must be an integer.

CALLING SEQUENCE: SUBARY(N,B(DV),C(DV),A(DV)

or ARYSUB(N,B(DV),C,A(DV))

The answer array may be overlayed into one of the input array areas.

MULTIPLICATION OPERATION

SUBROUTINE NAMES: MLTBLY or MPYFIX

PURPOSE:

To multiply a variable number of floating point or integer numbers respectively.

$$P = X1*X2*X3...*XN \quad , \quad N \geq 2$$

RESTRICTIONS:

Subroutine MLTPLY is for floating point numbers while subroutine MPYFIX is for integers.

CALLING SEQUENCE: MLTPLY(X1,X2,X3,...,XN,P)

or MPYFIX(X1,X2,X3,...,XN,P)

SUBROUTINE NAMES: MPYARY or ARYMPY

PURPOSE:

Subroutine MPYARY will multiply the corresponding elements of two arrays to form a third. Subroutine ARYMPY will multiply a constant value times each element of an array to form a new array. Their respective operations are:

$$Ai = Bi * Ci \quad , \quad i = 1,N$$

or

$$Ai = Bi * C \quad , \quad i = 1,N$$

RESTRICTIONS:

All data values to be operated on must be floating point numbers. The array length N must be an integer.

CALLING SEQUENCE:

MPYARY(N,B(DV),C(DV),A(DV))

or ARYMPY(N,B(DV),C,A(DV))

The answer array may be overlayed into one of the input array areas.

MULTIPLICATION OPERATION

SUBROUTINE NAMES: SCLDEP or SCLIND

PURPOSE:

These subroutines will multiply the dependent or independent variables of a doublet type interpolation array respectively. Their respective operations are:

$$A_i = X*A_i \quad , \quad i = 3,5,7,\ldots,N+1$$

$$\text{or} \quad A_i = X*A_i \quad , \quad i = 2,4,6,\ldots,N$$

RESTRICTIONS:

All values must be floating point. The arrays must contain the length integer count as the first value which must be even.

CALLING SEQUENCE: SCLDEP(A(IC),X)

or SCLIND(A(IC),X)

SUBROUTINE NAMES: CMPXMP or CMPYI

PURPOSE:

These subroutines will multiply two complex numbers or the corresponding elements of arrays of complex numbers. Their respective operations are:

$$A + iB = (C + iD)*(E + iF) \quad , \quad i = \sqrt{-1}$$

$$\text{or} \quad Aj + iBj = (Cj + iDj)*(Ej + iFj) \quad , \quad j = 1,N$$

RESTRICTIONS:

All numbers must be floating point except for N which must be an integer.

CALLING SEQUENCE: CMPXMP(C,D,E,F,A,B)

or CMPYI(N,C(DV),D(DV),E(DV),F(DV),A(DV),B(DV))

DIVISION OPERATION

SUBROUTINE NAMES: DIVIDE or DIVFIX

PURPOSE:

To perform a division of floating point or integer numbers respectively.

$$Q = Y/\Sigma\ Xi\ ,\ i = 1,2,3,\ldots,N\ ,\ N \geq 1$$

RESTRICTIONS:

Subroutine DIVIDE is for floating point numbers while DIVFIX is for integers.

CALLING SEQUENCE: DIVIDE(Y,X1,X2,X3,...,XN,Q)

or DIVFIX(Y,X1,X2,X3,...,XN,Q)

SUBROUTINE NAMES: DIVARY or ARYDIV

PURPOSE:

Subroutine DIVARY will divide the elements of one array into the corresponding elements of another array to produce a third array. Subroutine ARYDIV will divide each element of an array by a constant value to produce a new array. Their respective operations are:

$$Ai = Bi/Ci\ ,\ i = 1,N$$

or $$Ai = Bi/C\ ,\ i = 1,N$$

RESTRICTIONS:

All data values to be operated on must be floating point numbers. The array length N must be an integer.

CALLING SEQUENCE: DIVARY(N,B(DV),C(DV),A(DV))

or ARYDIV(N,B(DV),C,A(DV))

The answer array may be overlayed into one of the input array areas.

DIVISION OPERATION

SUBROUTINE NAMES: ARYINV or ARINDV

PURPOSE:

Subroutine ARYINV will invert each element of an array in its own location. Subroutine ARINDV will divide each element of an array into a constant value to forma a new array. Their respective operations are:

$$A_i = 1.0/A_i \quad , \quad i = 1,N$$
$$\text{or} \quad A_i = B/C_i \quad , \quad i = 1,N$$

RESTRICTIONS:

All data values must be floating point numbers. The array length N must be an integer.

CALLING SEQUENCE: ARYINV(N,A(DV))

or ARINDV(N,C(DV),B,A(DV))

(The ARINDV answer array may be overlayed into the input array area.)

SUBROUTINE NAMES: ADDINV or ADARIN

PURPOSE:

Subroutine ADDINV will calculate one over the sum of the inverses of a variable number of arguments. Subroutine ADARIN will calculate one over the sum of inverses of an array of values. These subroutines are useful for calculating the effective conductance of series conductors. Their respective operations are:

$$Y = 1.0/(1./X1 + 1./X2 + .. + 1./XN), \quad N \geq 2$$
$$\text{or} \quad Y = 1.0/\Sigma(1./X_i) \quad , \quad i = 1,2, \ldots , N$$

RESTRICTIONS:

All data values must be floating point numbers. The array length N must be an integer.

CALLING SEQUENCE: ADDINV(X1,X2,X3,...XN,Y)

or ADARIN(N,X(DV),Y)

DIVISION OPERATION

SUBROUTINE NAMES: CMPXDV or CDIVI

PURPOSE:

These subroutines will divide two complex numbers or the corresponding elements of arrays of complex numbers. Their respective operations are:

$$A + iB = (C + iD)/(E + iF) \quad , \quad j = \sqrt{-1}$$

$$\text{or} \quad Aj + iBj = (Cj + iDj)/(Ej + iFj) \quad , \quad j = 1,N$$

RESTRICTIONS:

All numbers must be floating point except for N which must be an integer.

CALLING SEQUENCE:

```
CMPXDV(C,D,E,F,A,B)
or  CDIVI(N,C(DV),D(DV),E(DV),F(DV),A(DV),B(DV))
```

INTEGER/FLOATING POINT CONVERSION

SUBROUTINE NAMES: FLØAT or FIX or INTRFC

PURPOSE:

Subroutine FLØAT will convert an integer to a floating point number. Subroutine FIX will convert a floating point number to an integer. Subroutine INTRFC will fracture a floating point number to yield the largest integer value possible and the remainder or fractional portion as a floating point number. Their respective operations are:

```
   X = N
or N = X
or N = X
   Y = N
   F = X-Y
```

RESTRICTIONS:

X and F arguments must address floating point values and the N argument address an integer.

CALLING SEQUENCE:

```
   FLØAT(N,X)
or FIX(X,N)
or INTRFC(X,N,F)
```

SIGN CONVERSION

SUBROUTINE NAMES: SETPLS or ARYPLS

PURPOSE:

SETPLS will set the sign positive for a variable number or arguments, while ARYPLS will set the sign positive for every data value in a specified length array.

RESTRICTIONS:

The values addressed may be either integers or floating point numbers. The number (N) of data values in the array must be specified as an integer.

CALLING SEQUENCE: SETPLS(A,B,C...)

or ARYPLS(N,A(DV))

where N may be a literal integer or the address of a location containing an integer and A(DV) addresses the first data value in the array.

SUBROUTINE NAMES: SETMNS or ARYMNS

PURPOSE:

SETMNS will set the sign negative for a variable number of arguments, while ARYMNS will set the sign negative for every data value in a specified length array.

RESTRICTIONS:

The values addressed may be either integers or floating point numbers. The number (N) of data value in the array must be specified as an integer.

CALLING SEQUENCE: SETMNS(A,B,C,...)

or ARYMNS(N,A(DV))

where N may be a literal integer or the address of a location containing an integer and A(DV) addresses the first data value in the array.

DISTRIBUTION OF ARRAY DATA

SUBROUTINE NAMES: SHFTV or SHFTVR or FLIP

PURPOSE:

Subroutine SHFTV will shift a sequence of data from one array to another. Subroutine SHFTVR will shift a sequence of data from one array and place it in another array in reverse order. Subroutine FLIP will reverse an array in its own array location. Their respective operations are:

```
        A(i) = B(i)              ,   i = 1,N
or      A(N-i+1) = B(i)          ,   i = 1,N
or      A(i)new = A(N-i+2)old    ,   i = 2,N+1
```

RESTRICTIONS:

The data values to be shifted or reversed in order may be anything. The N must be an integer.

CALLING SEQUENCE:

```
        SHFTV(N,B(DV),A(DV))
or      SHFTVR(N,B(DV),A(DV))
        FLIP(A(IC))
```

The answer array may not be overlayed into the input array.

SUBROUTINE NAME: GENARY

PURPOSE:

This subroutine will generate an array of equally incremented ascending values. The user must supply the minimum value, maximum value, number of values in the array to be generated and the space for the generated array.

RESTRICTIONS:

All numbers must be floating point.

CALLING SEQUENCE: GENARY(B(DV),A(DV))

where B(1) = minimum value
B(2) = maximum value
B(3) = length of array to be generated (floating point)

DISTRIBUTION OF ARRAY DATA

SUBROUTINE NAME: BLDARY

PURPOSE:

This subroutine will build an array from a variable number of arguments in the order listed. The operation performed is:

$$Ai = Xi \quad , \quad i = 1,N$$

RESTRICTIONS:

Data may be of any form. The subroutine obtains the integer array length N by counting the arguments.

CALLING SEQUENCE: BLDARY(A(DV),X1,X2,X3,...,XN)

SUBROUTINE NAME: BRKARY or BKARAD

PURPOSE:

These subroutines will distribute values from within an array to a variable number of arguments in the order listed. The first places the value into the location while the second adds it to what is in the location. Respective operations are:

$$Xi = Ai \quad , \quad i = 1,N$$

or

$$Xi = Xi + Ai \quad , \quad i = 1,N$$

RESTRICTIONS:

Floating point numbers must be used for BKARAD. The integer array length N is obtained by the routines by counting the number of arguments.

CALLING SEQUENCE: BRKARY(A(DV),X1,X2,X3,...,XN)

or BKARAD(A(DV),X1,X2,X3,...,XN)

SUBROUTINE NAMES: STØARY or ARYSTØ

PURPOSE:

These subroutines will place a value into or take a value out of a specific array location respectively. Their respective operations are:

$$Ai = X \quad , \quad i = N \quad , \quad N > 0$$

or

$$X = Ai \quad , \quad i = N \quad , \quad N > 0$$

RESTRICTIONS:

The value may be anything but N must be an integer.

CALLING SEQUENCE: STØARY(N,X,A(DV))

or ARYSTØ(N,X,A(DV))

DISTRIBUTION OF ARRAY DATA

SUBROUTINE NAMES: STFSEP or SCALE

PURPOSE:

Subroutine STFSEP will place a constant value into a variable number of locations. Subroutine SCALE will utilize a constant value to multiply a variable number of arguments, each having a location for the product. The respective operations are:

$$X_i = Y \quad , \quad i = 1,2,3,\ldots,N$$

or

$$X_i = Y*Z_i \quad , \quad i = 1,2,3,\ldots,N$$

RESTRICTIONS:

STFSEP may be used to move any desired value but SCALE can only be used for floating point numbers.

CALLING SEQUENCE: STFSEP(Y,X1,X2,X3,...,XN)

or SCALE(Y,Z1,X1,Z2,X2,...,ZN,XN)

SUBROUTINE NAMES: STFSEQ or STFSQS

PURPOSE:

Both subroutines will stuff a constant data value into a specified length array or group of sequential locations. STFSEQ expects the constant data value to be in the first array location while STFSQS requires it to be supplied as an additional argument. The respective operations performed are:

$$A_i = A1 \quad , \quad i = 2,N$$

or

$$A_i = B \quad , \quad i = 1,N$$

RESTRICTIONS:

N must be an integer but the constant data value may be integer, floating point or alpha-numeric.

CALLING SEQUENCE: STFSEQ(A(DV),N)

or STFSQS(B,N,A(DV))

DISTRIBUTION OF ARRAY DATA

SUBROUTINE NAMES: SLDARY or SLDARD

PURPOSE:

These subroutines are useful for updating fixed length interpolation arrays during a transient analysis. The array data values are moved back one or two positions, the first one or two values discarded and the last one or two values updated respectively. The "sliding array" thus maintained can then be used with standard interpolation subroutines to simulate transport delay phenomina. Their respective operations are:

	Ai = Ai +1	,	i = 2,N
and	Ai = X	,	i = N + 1
or	Ai = Ai + 2	,	i = 2,N-1
and	Ai = X and Ai + 1 = Y	,	i = N

RESTRICTIONS:

The addressed arrays must have the array integer count N as the first value. For SLDARD, N must be even.

CALLING SEQUENCE: SLDARY(X,A(IC))
SLDARD(X,Y,A(IC))

SUBROUTINE NAME: STØRMA

PURPOSE:

This subroutine is useful for constructing historical data arrays during a transient analysis. It can take the place of several STØARY calls. The operations are as follows:

```
A1(N) = X1
A2(N) = X2
A3(N) = X3
  :
  :
```

RESTRICTIONS:

N must be or reference an integer, the X's may be any value.

CALLING SEQUENCE: STØRMA(N,X1,A1(DV),X2,A2(DV),X3,A3(DV),...)

SINGLET/DOUBLET ARRAY GENERATION

SUBROUTINE NAMES: SPLIT or JØIN

PURPOSE:

These subroutines separate a doublet array into two singlet arrays or combine two singlet arrays into a doublet array respectively. Their respective operations are:

$$\begin{aligned} & Bi = A2i-1 && , && i = 1,N \\ & Ci = A2i && , && i = 1,N \\ \text{or} \quad & A2i-1 = Bi && , && i = 1,N \\ & A2i = Ci && , && i = 1,N \end{aligned}$$

RESTRICTIONS:

The arrays may contain any values but N must be an integer. N is the length of the B and C arrays and the A array must be of length 2N.

CALLING SEQUENCE: SPLIT(N,A(DV),B(DV),C(DV))

or JOIN(N,B(DV),C(DV),A(DV))

SUBROUTINE NAME: SPREAD

PURPOSE:

This subroutine applies interpolation subroutine D1D1DA to two singlet arrays to obtain an array of dependent variables versus an array of independent variables. It is extremely useful for obtaining singlet arrays of various dependent variables with a corresponding relationship to one singlet independent variable array. The dependent variable arrays thus constructed can then be operated on by array manipulation subroutines in order to form composite or complex functions. Doublet arrays can first be separated with subroutine SPLIT and later reformed with subroutine JØIN.

RESTRICTIONS:

All data values must be floating point except N which must be the integer length of the array to be constructed. The arrays fed into D1D1DA for interpolation must start with the integer count. X is for independent and Y is for dependent. I is for input and Ø is for output.

CALLING SEQUENCE: SPREAD(N,X(IC),Y(IC),XI(DV),YØ(DV))

COMPARISON OPERATION

SUBROUTINE NAMES: MAXDAR or MXDRAL

PURPOSE:

These subroutines will obtain the absolute maximum difference between corresponding elements of two arrays of equal length N. The array values must be floating point numbers. The operation performed is

$$D = \left| Ai - Bi \right|_{max} , \quad i = 1,N$$

Subroutine MXDRAL also locates the position P between 1 and N where the maximum occurs.

RESTRICTIONS:

The N argument must be an integer. The D and P arguments are returned as floating point numbers.

CALLING SEQUENCE: MAXDAR(N,A(DV),B(DV),D)

or MXDRAL(N,A(DV),B(DV),D,P)

A.4 INTERPOLATION/EXTRAPOLATION SUBROUTINES

LAGRANGIAN INTERPOLATION

SUBROUTINE NAMES: LAGRAN or LGRNDA

PURPOSE:

These subroutines perform Lagrangian interpolation of up to order 50. The first requires one doublet array of X, Y pairs while the second requires two singlet arrays, one of X's and the other of Y's. They contain an extrapolation feature such that if the X value falls outside the range of the independent variable the nearest dependent Y variable value is returned and no error is noted.

$$Y = Pn\ (X) = \sum_{k=0}^{n} Yk \prod_{\substack{i=0 \\ i \neq k}}^{n} \frac{X - Xi}{Xk - Xi} \quad , n = 1,2,3,\ldots,50max.$$

RESTRICTIONS:

All values must be floating point except N which is the order of interpolation plus one and must be an integer. The independent variable values must be in ascending order.

CALLING SEQUENCE: LAGRAN(X,Y,A(IC),N)

or LGRNDA(X,Y,AX(IC),AY(IC),Y)

NOTE:

A doublet array is formed as follows:

IC,X1,Y1,X2,Y2,X3,Y3,...,XN,YN
where IC = 2*N (set by program)

and singlet arrays are formed as follows:

IC,X1,X2,X3,...,XN
IC,Y1,Y2,Y3,...,YN
and IC = N (set by program)

LINEAR INTERPOLATION - SINGLE VARIABLE

SUBROUTINE NAMES: D1DEG1 or D1D1DA

PURPOSE:

These subroutines perform single variable linear interpolation on doublet or singlet arrays respectively. They are self-contained subroutines that are called upon by virtually all other linear interpolation subroutines.

RESTRICTIONS:

All values must be floating point numbers. The X independent variable values must be in ascending order.

CALLING SEQUENCE: D1DEG1(X,A,(IC),Y)
or D1D1DA(X,AX(IC),AY(IC),Y)

SUBROUTINE NAMES: D1D1WM or D11MDA

PURPOSE:

These subroutines perform single variable linear interpolation by calling on D1DEG1 or D1D1DA respectively. However, the interpolated answer is multiplied by the value addressed as Z prior to being returned as Y.

RESTRICTIONS:

Same as D1DEG1 or D1D1DA and Z must be a floating point number.

CALLING SEQUENCE: D1D1WM(X,A(IC),Z,Y)

or D11MDA(X,AX(IC),AY(IC),Z,Y)

SUBROUTINE NAMES: D1MDG1 or D1M1DA

PURPOSE:

These subroutines use the arithmetic mean of two input values as the independent variable for linear interpolation. They require a doublet or two singlet arrays respectively.

RESTRICTIONS:

See D1DEG1 or D1D1DA as they are called on respectively.

CALLING SEQUENCE: D1MDG1(X1,X2,A(IC),Y)

or D1M1DA(X1,X2,AX(IC),AY(IC),Y)

LINEAR INTERPOLATION - SINGLE VARIABLE

SUBROUTINE NAMES: D1M1WM or D1M1MD

PURPOSE:

These subroutines use the arithmetic mean of two input values as the independent variable for linear interpolation. The interpolated answer is multiplied by the Z value prior to being returned as Y.

RESTRICTIONS:

Same as D1MDG1 or D1M1DA and Z must be a floating point number.

CALLING SEQUENCE: D1M1WM(X1,X2,A(IC),Z,Y) or D1M1MD(X1,X2,AX(IC),AY(IC),Z,Y)

SUBROUTINE NAMES: D1DG1I or D1D1IM or D1D1MI

PURPOSE:

These subroutines perform single variable linear interpolation on an array of X's to obtain an array of Y's. D1D1IM multiplies all interpolated values by a constant Z value while D1D1MI allows a unique Z value for each X value. They all call on D1DEG1.

RESTRICTIONS:

The number of input X's must be supplied as the integer N and agree with the number of Y and Z locations where applicable. Z values must be floating point numbers.

CALLING SEQUENCE: D1DG1I(N,X(DV),A(IC),Y(DV))
or D1D1IM(N,X(DV),A(IC),Z,Y(DV))
or D1D1MI(N,X(DV),A(IC),Z(DV),Y(DV))

SUBROUTINE NAMES: D11DAI or D11DIM or D11MDI

PURPOSE:

These subroutines are virtually identical to D1DG1I, D1D1IM and D1D1MI respectively. The difference is that they require singlet arrays for interpolation and call on D1D1DA.

RESTRICTIONS:

Same as D1DG1I, D1D1IM and D1D1MI.

CALLING SEQUENCE: D11DAI(N,X(DV),AX(IC),AY(IC),Y(DV))
or D11DIM(N,X(DV),AX(IC),AY(IC),Z,Y(DV))
or D11MDI(N,X(DV),AX(IC),AY(IC),Z(DV),Y(DV))

LINEAR INTERPOLATION - SINGLE VARIABLE/TWO SINGLE VARIABLES

SUBROUTINE NAMES: D1IMD1 or D1IMWM or D1IMIM

PURPOSE:

These are indexed subroutines which use the arithmetic mean of two input values as the independent variable for linear interpolation. The string of answers (Y) produced are left as is (D1IMD1), are all multiplied by a single factor (D1IMWM), or each answer is multiplied by a separate factor.

RESTRICTIONS:

The interpolation array addressed must have an even number of input values and the independent variables must be in ascending order. These routines call upon D1D1WM. N is the number of times the operation is to be performed.

CALLING SEQUENCE:

```
D1IMD1(N,X1(DV),X2(DV),A,Y(DV))
or  D1IMWM(N,X1(DV),X2(DV),A,Z,Y(DV))
or  D1IMIM(N,X1(DV),X2(DV),A,Z(DV),Y(DV)
```

LINEAR INTERPOLATION - TWO SINGLE VARIABLES

SUBROUTINE NAMES: CVQ1HT or CVQ1WM

PURPOSE:

These subroutines perform two single variable linear interpolations. The interpolation arrays must have the same independent variable X and dependent variables of, let's say, R(X) and S(X). Additional arguments of Y, Z and T complete the data values. The post interpolation calculations are respectively:

$$Y = S(X)*(R(X)-T)$$

or $$Y = Z*S(X)(R(X)-T)$$

RESTRICTIONS:

Interpolation arrays must be of the doublet type and have a common independent variable. All values must be floating point numbers.

CALLING SEQUENCE:

```
CVQ1HT(X,AR(IC),AS(IC),T,Y)
or  CVQ1WM(X,AR(IC),AS(IC),T,Z,Y)
```

SUBROUTINE NAMES: VARCSM or VARCCM or VARC1 or VARC2

PURPOSE: These are linear interpolation subroutines carried over from CINDA-3G. Mnemonic options utilized in the Node Data block caused insertion of these calls into the Variables 1 block. This does not apply for the SINDA program but the routines remain as they could be called directly by a user. The routines are similar in that the C argument is a function of the T argument which is the independent variable for interpolation from the doublet array A argument, answer to which is multiplied by the factor F argument. Where two A's and F's are referenced in the same call, separate interpolations and multiplications are performed and the answers summed.

RESTRICTIONS: VARC1 and VARC2 reference only one A argument for interpolation, the other A-F position arguments are multiplied together to form their contribution to the answer.

CALLING SEQUENCE:

```
VARCSM (T, C, A(IC), F)
VARCCM (T, C, A1(IC), F1, A2(IC), F2)
VARC1  (T, C, 1.34, F1, A2(IC), F2)
VARC2  (T, C, A1(IC), F1, 2.87, F2)
```

SUBROUTINE NAMES: VARGSM or VARGCM or VARG1 or VARG2

PURPOSE: As above, these routines are carried over from the CINDA-3G program except that they pertained to the conductor block. The mean temperature of the two T arguments is used as the independent variable for interpolation when VARGSM is called except if the F argument is negative, in which case the T1 argument is used. The other three routines use the T1 for A1 and/or T2 for A2 to obtain two partial values (as described above) which are then combined as one over the sum of the inverses:

$$G = 1.0/(1.0/G1 + 1.0/G2)$$

RESTRICTIONS: The A arguments must reference the integer count of of doublet interpolation arrays.

CALLING SEQUENCE:

```
VARGSM (G, T1, T2, A(IC), F)
VARGCM (G, T1, T2, A1(IC), F1, A2(IC), F2)
VARG1  (G, T1, T2, 4.78, F1, A2(IC), F2)
VARG2  (G, T1, T2, A1(IC), F1, 7.93, F2)
```

PARABOLIC INTERPOLATION - SINGLE VARIABLE

SUBROUTINE NAMES: D1DEG2 or D1D2DA

PURPOSE:

These subroutines perform single variable parabolic interpolation. The first requires a double array of X, Y pairs while the second requires singlet arrays of X and Y. values. They call on subroutines LAGRAN and LGRNDA respectively.

RESTRICTIONS:

See LAGRAN or LGRNDA respectively.

CALLING SEQUENCE: D1DEG2(X,A(IC),Y)

or D1D2DA(X,AX(IC),AY(IC),Y)

SUBROUTINE NAMES: D1D2WM or D12MDA

PURPOSE:

These subroutines perform single variable parabolic interpolation by calling on LAGRAN or LGRNDA respectively. However, the interpolated answer is multiplied by the value addressed as Z prior to being returned as Y.

RESTRICTIONS:

Same as LAGRAN or LGRNDA and Z must be a floating point number.

CALLING SEQUENCE: D1D2WM(X,A(IC),Z,Y)

or D12MDA(X,AX(IC),AY(IC),Z,Y)

PARABOLIC INTERPOLATION - SINGLE VARIABLE

SUBROUTINE NAMES: D1MDG2 or D1M2DA

PURPOSE:

These subroutines use the arithmetic mean of two input values as the independent variable for parabolic interpolation. They require a doublet or two singlet arrays respectively.

RESTRICTIONS:

See LAGRAN or LGRNDA as they are called on respectively.

CALLING SEQUENCE: D1MDG2(X1,X2,A(IC),Y)

or D1M2DA(X1,X2,AX(IC),AY(IC),Y)

SUBROUTINE NAMES: D1M2WM or D1M2MD

PURPOSE:

These subroutines use the arithmetic mean of two input values as the independent variable for parabolic interpolation. The interpolated answer is multiplied by the Z value prior to being returned as Y.

RESTRICTIONS:

Same as D1MDG2 or D1M2DA and Z must be a floating point number.

CALLING SEQUENCE: D1M2WM(X1,X2,A(IC),Z,Y)

or D1M2MD(X1,X2,AX(IC),AY(IC),Z,Y)

SUBROUTINE NAMES: D11CYL or DA11CY

PURPOSE: These subroutines reduce core storage requirements for cyclical interpolation arrays. The arrays need cover one period only, and the period (PR) must be specified as the first argument. Linear interpolation is performed, and the independent variable must be in ascending order.

RESTRICTIONS: All values must be floating point. Subroutine INTRFC is called on by both D11CYL and DA11CY, then D1DEG1 or D1D1DA respectively.

CALLING SEQUENCE: D11CYL(PR,X,A(IC),Y)

or DA11CY(PR,X,AX(IC),AY(IC),Y)

SUBROUTINE NAMES: D12CYL or DA12CY

PURPOSE: These subroutines are virtually identical to D11CYL and DA11CY except that parabolic interpolation is performed.

RESTRICTIONS: See D11CYL and DA11CY. Subroutines LAGRAN and LGRNDA respectively are called on.

CALLING SEQUENCE: D12CYL(PR,X,A(IC),Y)

or DA12CY(PR,X,AX(IC),AY(IC),Y)

SUBROUTINE NAMES: D11MCY or DA11MC

PURPOSE: These subroutines are virtually identical to D11CYL and DA11CY except that the interpolation is multiplied by the floating point Z value prior to being returned as Y.

RESTRICTIONS: Call on subroutines D1DEG1 and D1D1DA respectively.

CALLING SEQUENCE: D11MCY(PR,X,A(IC),Z,Y)

or DA11MC(PR,X,AX(IC),AY(IC),Z,Y)

SUBROUTINE NAMES: D12MCY or DA12MC

PURPOSE: These subroutines are virtually identical to D11MCY and DA11MC except that parabolic interpolation is performed.

RESTRICTIONS: Calls on subroutines LAGRAN and LGRNDA respectively.

CALLING SEQUENCE: D12MCY(PR,X,A(IC),Z,Y)

or DA12MC(PR,X,AX(IC),AY(IC),Z,Y)

POINT SLOPE INTERPOLATION

SUBROUTINE NAMES: GSLØPE

PURPOSE:

This subroutine will generate a slope array so that point slope interpolation subroutines can be used instead of standard linear interpolation subroutines. The user must address two singlet type arrays and a singlet slope array will be produced.

RESTRICTIONS:

The X independent variable array must be in ascending order. All arrays must be of equal length and contain floating point numbers.

CALLING SEQUENCE:

```
GSLØPE(AX(IC),AY(IC),AS(IC))
```

SUBROUTINE NAMES: PSINTR or PSNTWM

PURPOSE:

These subroutines perform linear interpolation and require arrays of the Y points and slopes which correspond to the independent variable X array. All values must be floating point numbers. PSNTWM multiplies the interpolated answer by Z prior to returning it as Y.

RESTRICTIONS:

The independent X and dependent Y and slope arrays must be of equal length.

CALLING SEQUENCE:

```
PSINTR(X,AX(IC),AY(IC),AS(IC),Y)
```

or

```
PSNTWM(X,AX(IC),AY(IC),AS(IC),Z,Y)
```

BIVARIATE ARRAY FORMAT $Z = f(X,Y)$

Bivariate arrays must be rectangular, full and input in the following row order:

```
IC,N ,X 1,X 2,X 3, . . . , X N
   Y1,Z11,Z12,Z13, . . . , Z1N
   Y2,Z21,Z22,Z23, . . . , Z2N
   .                       .
   .                       .
   .                       .
   YM,ZM1,ZM2,ZM3, . . . , ZMN
```

where N is the integer number of X variables. All other values must be floating point numbers, and the X and Y values must be in ascending order.

SUBROUTINE NAMES: BVSPSA or BVSPDA

PURPOSE: These subroutines use an input Y argument to address a bivariate array and pull off a singlet array of Z's corresponding to the X's or pull off a doublet array of X, Z values, respectively. The integer count for the constructed arrays must be exactly N or 2*N respectively. To use the singlet array for an interpolation call the X array can be reached by addressing the N in the bivariate array.

RESTRICTIONS: As stated above, and all values must be floating point.

CALLING SEQUENCE: BVSPSA(Y,BA(IC),AZ(IC))

or BVSPDA(Y,BA(IC),AXZ(IC))

SUBROUTINE NAMES: BVTRN1 or BVTRN2

PURPOSE: These subroutines construct a bivariate array of Y's versus X and Z from an input bivariate array of Z's versus X and Y. BVTRN1 should be used when the Z values increase with increasing Y values and BVTRN2 when the Z values decrease with increasing Y values.

RESTRICTIONS: The user must appropriately place the X and Z values and spaces for Y's in the array to be constructed. These subroutines will fill the Y spaces. The new array can differ in size from the old. Subroutine D1DEG1 is called and its linear extrapolation feature applies.

CALLING SEQUENCE: BVTRN1(BAØ(IC),BAN(IC))

or BVTRN2(BAØ(IC),BAN(IC))

SUBROUTINE NAMES: D2DEG1 or D2DEG2

PURPOSE: These subroutines perform bivariate linear and parabolic interpolation respectively. The arrays msut be formated as shown for Bivariate Array Format.

RESTRICTIONS: For D2DEG1 , N≥2,M≥2 See Bivariate
for D2DEG2 , N≥3,M≥3 Array Format

CALLING SEQUENCE: D2DEG1(X,Y,BA(IC),Z)

or D2DEG2(X,Y,BA(IC),Z)

SUBROUTINE NAMES: D2D1WM or D2D2WM

PURPOSE: These subroutines perform bivariate linear or parabolic interpolation by calling on D2DEG1 or D2DEG2 respectively. The interpolated answer is multiplied by the W value prior to being returned as Z.

RESTRICTIONS: Same as D2DEG1 or D2DEG2 and W must be a floating point value.

CALLING SEQUENCE: D2D1WM(C,Y,BA(IC),W,Z)

or D2D2WM(X,Y,BA(IC),W,Z)

SUBROUTINE NAMES: D2MXD1 or D2MXD2

PURPOSE: These subroutines are virtually identical to D2DEG1 and D2DEG2 except that the arithmetic mean of two X values is used as the X independent variable for interpolation.

RESTRICTIONS: Same as D2DEG1 or D2DEG2.

CALLING SEQUENCE: D2MXD1(X1,X2,Y,BA(IC),Z)

or D2MXD2(X1,X2,Y,BA(IC),Z)

SUBROUTINE NAMES: D2MX1M or D2MX2M

PURPOSE: These subroutines are virtually identical to D2D1WM and D2D2WM except that the arithmetic mean of two X values is used as the X independent variable for interpolation.

RESTRICTIONS: Same as D2D1WM and D2D2WM.

CALLING SEQUENCE: D2MX1M(X1,X2,Y,BA(IC),W,Z)

or D2MX2M(X1,X2,Y,BA(IC),W,Z)

TRIVARIATE INTERPOLATION

TRIVARIATE ARRAY FORMAT T = f(X,Y,Z)

Trivariate arrays may be thought of as two or more bivariate arrays, each bivariate array a function of a third independent variable Z. Trivariate arrays must be input in row order and be constructed as follows:

```
IC,NX1,NY1,Z1,X 1,X 2,X 3, . . . , X N
           Y1,T11,T12,T13, . . . , T1N
           Y2,T21,T22,T23, . . . , T2N
            .  .  .  .  .  .  .  .  .
           YM,TM1,TM2,TM3, . . . , TMN
  NX2,NY2,Z2,X 1,X 2,X 3, . . . , X J
           Y1,T11,T12,T13, . . . , T1J
           Y2,T21,T22,T23, . . . , T2J
            .  .  .  .  .  .  .  .  .
           YK,TK1,TK2,TK3, . . . , TKJ
  NX3,NY3,Z3, . . . . . .
            .  .  .  .  .  .  .  .  .
            .  .  .  .  .  .  .  .  .
```

The trivariate array may consist of as many bivariate "sheets" as desired. The number of X and Y values in each sheet must be specified as integers (NX-NY). The "sheets" must be rectangular and full but need not be identical in size.

SUBROUTINE NAMES: D3DEG1 or D3D1WM

PURPOSE:

These subroutines perform trivariate linear interpolation. The interpolation array must be constructed as shown for Trivariate Array Format. Subroutine D2DEG1 is called on which calls on D1DEG1. Hence, the linear extrapolation feature of these routines applies. Subroutine D3D1WM multiplies the interpolated answer by F prior to returning it as T.

RESTRICTIONS:

See Trivariate Array Format. F must be a floating point value.

CALLING SEQUENCE: D3DEG1(X,Y,Z,TA(IC),T)

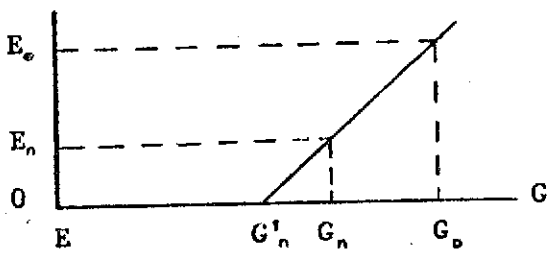
or D3D1WM(X,Y,Z,TA(IC),F,T)

LINEAR EXTRAPOLATION

SUBROUTINE NAME: ITRATE

PURPOSE:

Given two old guesses and their corresponding errors, this routine linearly extrapolates a new guess on the basis of zero error.



The new guess and error are positioned in the old locations and the extrapolated new guess is returned in the new guess location.

RESTRICTIONS:

If the error function being plotted has changes of slope, the user must insure that his guesses are quite accurate or divergence will be assured.

CALLING SEQUENCE:

ITRATE(EØ,GØ,EN,GN)

A.5 MATHEMATICAL SOLUTION SUBROUTINES

Area Integration

SMPINT, TRPZD	Performs area integration by Simpson's rule and trapezoidal rule using equal increments	A.5-3 "
TRPZDA	Performs area integration by the trapezoidal rule with non uniform increments	"

Functional Evaluation

CINSIN, SINARY	Obtains the sine function of an angle	A.5-4 "
CINCØS, CØSARY	Obtains the cosine function of an angle or array of angles	"
CINTAN, TANARY	Obtains the tangent functions of an angle or array of angles	" "
ARCSIN, ASNARY	Obtains the angle corresponding to a sine function value or array of sine values	A.5-5 "
ARCCØS, ACSARY	Obtains the angle corresponding to a cosine function value or array of cosine values	" "
ARCTAN, ATNARY	Obtains the angle corresponding to a tangent function value of array of tangent values	" "
EXPNTL, ARYEXP, EXPARY	Performs exponential operations	A.5-6
LØGT, LØGTAR	Obtains the base 10 log function of a number or array of numbers	" "
LØGE, LØGEAR	Obtains the base e log function of a number or array of numbers	"

Roots

SQRØØT, SQRØTI	Obtains the square root of a number or array of numbers	A.5-7 "
CMPXSR, CSQRI	Obtains the complex square root of a complex number or array of complex numbers	" "
NEWTRT, NEWRT4	Utilizes Newton's method to obtain one root of a cubic or quartic equation	A.5-8 "

Polynomial/Simultaneous Linear Equations

PLYNML, PLYARY, PLYAWM	Calculates the value of the dependent variable for an Nth order polynomial	A.5-9 " "
SIMEQN	Solves a set of linear equations (10 or less) by the factorized inverse method	"

Curve Fit/Temperature Derivative

LSTSQU	Performs a least squares curve fit to an arbitrary number of X,Y pairs to yield a polynomial equation of up to order 10	A.5-10
TDØT	Calculates the time point temperature derivatives for diffusion nodes	"

AREA INTEGRATION

SUBROUTINE NAMES: SMPINT or TRPZD

PURPOSE:

These subroutines perform area integrations by Simpson's rule and the trapezoidal rule respectively. Simpson's rule requires that an odd number of points be supplied. If an even number of points is supplied, SMPINT will apply the trapezoidal rule to the last incremental area but Simpson's rule elsewhere. The respective operations are:

$$A = DX*(Y1+4Y2+2Y3+4Y4+...+YN)/3$$

or $$A = DX*(Y1+2Y2+2Y3+2Y4+...+YN)/2$$

RESTRICTIONS:

The DX increment must be uniform between all the Y points. All values must be floating point except N which must be an integer.

CALLING SEQUENCE: SMPINT(N,DX,Y(DV),A)

or TRPZD(N,DX,Y(DV),A)

SUBROUTINE NAME: TRPZDA

PURPOSE:

This subroutine performs area integration by the trapezoidal rule. It should be used where the DX increment is not uniform between the Y values but the corresponding X value for each Y value is known. The operation performed is as follows:

$$A = \frac{1}{2} \Sigma \ (Xi-Xi-1)*(Yi+Yi-1) \quad , \quad i = 2,N$$

RESTRICTIONS:

All values must be floating point numbers except the array length N which must be an integer.

CALLING SEQUENCE: TRPZDA(N,X(DV),Y(DV),A)

SUBROUTINE NAMES: CINSIN or SINARY

PURPOSE:

These subroutines obtain the sine function of an angle or array of angles. Their respective operations are:

A = sine (B)
or Ai = sine (Bi) , i = 1,N

RESTRICTIONS:

All angles must be in radians. All values must be floating point numbers except N which must be an integer.

CALLING SEQUENCE: CINSIN(B,A)

or SINARY(N,B(DV),A(DV))

SUBROUTINE NAMES: CINCØS or CØSARY

PURPOSE:

These subroutines obtain the cosine function of an angle or array of angles. Their respective operations are:

A = cosine (B)
or Ai = cosine (Bi) , i = 1,N

RESTRICTIONS:

All angles must be in radians. All values must be floating point numbers except the array length N which must be an integer.

CALLING SEQUENCE: CINCØS(B,A)

or COSARY(N,B(DV),A(DV))

SUBROUTINE NAMES: CINTAN or TANARY

PURPOSE:

These subroutines obtain the tangent function of an angle or array of angles. Their respective operations are:

A = tangent (B)
or Ai = tangent (Bi) , i = 1,N

RESTRICTIONS:

All angles must be in radians. All values must be floating point numbers except the array length N which must be an integer.

CALLING SEQUENCE: CINTAN(B,A)

or TANARY(N,B(DV),A(DV))

FUNCTIONAL EVALUATION

SUBROUTINE NAMES: ARCSIN or ASNARY

PURPOSE:

These subroutines obtain the angle corresponding to a sine function value or array of sine values. Their respective operations are:

$$A = \text{sine}^{-1}(B)$$
$$\text{or} \quad Ai = \text{sine}^{-1}(Bi) \quad , \quad i = 1,N$$

RESTRICTIONS:

The angles are returned in radians with the following limits, $-\pi/2 \leq A \leq \pi/2$. All values must be floating point except for the array length N which must be an integer.

CALLING SEQUENCE: ARCSIN(B,A) or ASNARY(N,B(DV),A(DV))

SUBROUTINE NAMES: ARCCØS or ACSARY

PURPOSE:

These subroutines obtain the angle corresponding to a cosine function value or array of cosine values. Their respective operations are:

$$A = \text{cosine}^{-1}(B)$$
$$\text{or} \quad Ai = \text{cosine}^{-1}(Bi) \quad , \quad i = 1,N$$

RESTRICTIONS:

The angles are returned in radians with the following limits, $0 \leq A \leq \pi$. All values must be floating point numbers except for the array length N which must be an integer.

CALLING SEQUENCE: ARCCØS(B,A) or ACSARY(N,B(DV),A(DV))

SUBROUTINE NAMES: ARCTAN or ATNARY

PURPOSE:

These subroutines obtain the angle corresponding to a tangent function value of array of tangent values. Their respective operations are:

$$A = \text{tangent}^{-1}(B)$$
$$\text{or} \quad Ai = \text{tangent}^{-1}(Bi) \quad , \quad i = 1,N$$

RESTRICTIONS:

The angles are returned in radians with the following limits, $-\pi/2 < A < \pi/2$ All values must be floating point numbers except the array length N which must be an integer.

CALLING SEQUENCE: ARCTAN(B,A) or ATNARY(N,B(DV),A(DV))

SUBROUTINE NAMES: EXPNTL or ARYEXP or EXPARY

PURPOSE:

These subroutines perform an exponential operation. Their respective operations are:

$$A = B^C$$
$$\text{or} \quad Ai = Bi^C \quad , \quad i = 1,N$$
$$\text{or} \quad Ai = Bi^{Ci} \quad , \quad i = 1,N$$

RESTRICTIONS:

All values must be positive floating point numbers except N which must be an integer.

CALLING SEQUENCE: EXPNTL(C,B,A)

or ARYEXP(N,C,B(DV),A(DV))

or EXPARY(N,C(DV),B(DV),A(DV))

SUBROUTINE NAMES: LØGT or LØGTAR

PURPOSE:

These subroutines obtain the base 10 log function of a number or array of numbers. Their respective operations are:

$$A = \log_{10}(B)$$
$$\text{or} \quad Ai = \log_{10}(Bi) \quad , \quad i = 1,N$$

RESTRICTIONS:

All values must be positive floating point numbers except N which must be an integer.

CALLING SEQUENCE: LØGT(B,A)

or LØGTAR(N,B(DV),A(DV))

SUBROUTINE NAMES: LØGE or LØGEAR

PURPOSE:

These subroutines obtain the base e log function of a number or array of numbers. Their respective operations are:

$$A = \log_e(B)$$
$$\text{or} \quad Ai = \log_e(Bi) \quad , \quad i = 1,N$$

RESTRICTIONS:

All values must be positive floating point numbers except N which must be an integer.

CALLING SEQUENCE: LØGE(B,A)

or LØGEAR(N,B(DV),A(DV))

ROOTS

SUBROUTINE NAMES: SQRØØT or SQRØTI

PURPOSE:

These subroutines obtain the square root of a number or array of numbers respectively. Their respective operations are:

$$A = +\sqrt{B}$$

or $$Ai = +\sqrt{Bi} \quad , \quad i = 1,N$$

RESTRICTIONS:

The A and B values must be floating point numbers. The N must be an integer.

CALLING SEQUENCE: SQRØØT(B,A)

or SQRØTI(N,B(DV),A(DV))

SUBROUTINE NAMES: CMPXSR or CSQRI

PURPOSE:

These subroutines obtain the complex square root of a complex number or an array of complex numbers respectively. Their respective operations are:

$$A + iB = \sqrt{C + iD} \quad , \quad i = \sqrt{-1}$$

or $$Aj + iBj = \sqrt{Cj + iDj} \quad , \quad j = 1,N$$

RESTRICTIONS:

All numbers must be floating point except N which must be an integer.

CALLING SEQUENCE: CMPXSR(C,D,A,B)

or CSQRI(N,C(DV),D(DV),A(DV),B(DV))

ROOTS

SUBROUTINE NAMES: NEWTRT or NEWRT4

PURPOSE:

These subroutines utilize Newton's method to obtain one root of a cubic or quartic equation respectively. The root must be in the neighborhood of the supplied initial guess and up to 100 iterations are performed in order to obtain an answer within the specified tolerance. If the tolerance is not met, an answer of 10^{38} is returned. The respective equations are:

$$f(X) = A1+A2*X+A3*X^2+A4*X^3 = 0.0 \pm T$$

or

$$g(X) = A1+A2*X+A2*X^2+A4*X^3+A5*X^4 = 0.0 \pm T$$

where X starts as the initial guess RI and finishes as the final answer RF. T is the tolerance.

RESTRICTIONS:

All data values must be floating point numbers.

CALLING SEQUENCE: NEWTRT(A(DV),T,RI,RF)

or NEWRT4(A(DV),T,RI,RF)

POLYNOMIAL/SIMULTANEOUS LINEAR EQUATIONS

SUBROUTINE NAMES: PLYNML or PLYARY or PLYAWM

PURPOSE:

These subroutines calculate Y from the following polynomial equation:

$$Y = A1+A2*X+A3*X^2+A4*X^3+ \ldots +AN+1*X^N$$
$$Z = Y*W$$

The number of terms is variable but all the A coefficients must be input no matter what their value.

RESTRICTIONS:

All values must be floating point numbers except for the degree of polynomial N which must be integer.

CALLING SEQUENCE: PLYNML(X,A1,A2,A3,...,AN,Y)

or PLYARY(N,X,A(DV),Y)

or PLYAWM(N,X,A(DV),W,Z)

SUBROUTINE NAME: SIMEQN

PURPOSE:

This subroutine solves a set of up to 10 linear simultaneous equations by the factorized inverse method. The problem size and all input and output values are communicated as a single specially formatted positive input array. The array argument must address the matrix order (N) which is input by the user. The first data value must be the integer order of the set (or size of the square matrix) followed by the coefficient matrix [A] in column order, the boundary vector $\{B\}$ and space for the solution of vector $\{S\}$.

$$[A] \quad \{S\} = \{B\}$$

RESTRICTIONS:

The integer count and matrix size must be integers, all other values must be floating point. The coefficient matrix is not modified by SIMEQN. Hence, changes to $\{B\}$ only allow additional solutions to be easily obtained.

CALLING SEQUENCE: SIMEQN(A(N))

where the array is formatted exactly as follows:

IC,N,A(1,1),A(1,2),...A(N,N),B1,...,BN,S1,...,SN

CURVE FIT/TEMPERATURE DERIVATIVE

SUBROUTINE NAME: LSTSQU

PURPOSE:

This subroutine performs a least squares curve to fit to an arbitrary number of X, Y pairs to yield a polynomial equation of up to order 10. Rather than using a double precision matrix inverse, this subroutine calls on the subroutine SIMEQN to obtain a simultaneous solution.

RESTRICTIONS:

All values must be floating point numbers except N and M which must be integers. N is the order of the polynominal desired and is one less than the number of coefficients desired. M is the array length of the independent X or dependent Y values.

CALLING SEQUENCE: LSTSQU(N, M,X(DV),Y(DV),A(DV))

*This subroutine requires 2*M dynamic storage core locations.

SUBROUTINE NAME: TDØT

PURPOSE:

This subroutine allows the user to calculate the time point temperature derivatives for diffusion nodes. The single argument must address an array with as many locations as there are diffusion nodes; the answers are returned in relative order. The routine utilizes the pseudo-compute sequence to calculate the time point net q into the nodes and then divides by the nodal capacitances. Consequently, the user may multiply the temperature derivatives by the nodal capacitance to obtain the nodal net q.

RESTRICTIONS:

Do not call this subroutine from Variables 1. The long pseudo-compute sequence is required.

CALLING SEQUENCE: TDØT(A(DV))

Input Format

Unless otherwise noted, the matrices require input as positive numbered arrays with integer number of rows and columns as the first two data values followed by floating point element values in row order.

EFASN	Generates the arcsine of each element of matrix [A] and places it in the corresponding location of another matrix [Z]	A.6-7
EFCØS	Generates the cosine of each element of matrix [A] and places it in the corresponding location of another matrix [Z]	"
EFACS	Generates the arcosine of each element of matrix [A] and places it in the corresponding location of another matrix [Z]	"
EFTAN	Generates the tangent of each element of matrix [A] and places it in the corresponding location of another matrix [Z]	"
EFATN	Generates the arctangent of each element of matrix [A] and places it in the corresponding location of another matrix [Z]	"
EFABS	Generates the absolute value of each matrix [A] element	A.6-8
EFLØG	Generates the natural log of each matrix [A] element	"
EFSQR	Generates the square root of each matrix [A] element	"
EFEXP	Generates the exponential of each matrix [A] element	"
EFPØW	Generates the power of each matrix [A] element	"
ADDALP	Adds a constant to every element in a matrix	A.6-9
ALPHAA	Multiplies every element in a matrix by a constant	"
MATRIX	Allows a constant to replace a specific matrix element	"
SCALAR	Allows a specific matrix element to be placed into a constant location	"
MATADD	Adds a constant to a specific matrix element	"

Matrix Operations/Solutions

INVRSE	Inverts a square matrix	A.6-10
MULT	Multiplies two conformable matrices	"
TRANS	Forms the transpose [Z] from matrix [A]	"
AABB	Sums two scaled matrices	A.6-11
BTAB	Performs the matrix operation $[B]^t$ [A][B]	"
BABT	Performs the matrix operation [B][A]$[B]^t$	"
DISAS	Allows a user to operate on matrices in a partitioned manner by disassembling a submatrix [Z[from a parent matrix [A]	A.6-12
ASSMBL	Allows a user to operate on matrices in a partitioned manner by assembling a submatrix [Z] into a parent matrix [A]	"

CØLMLT, RØWMLT	Multiplies each element in a column or row of matrix [A] by its corresponding element from the diagonal matrix [V] which is stored as a vector	A.6-12
SHIFT	Moves an entire matirx as is from one location to another	A.6-13
REFLCT	Moves an entire matrix with the order of the column elements reversed from one location to another	"
SHUFL	Allows the user to reorder the size of a matrix of a matrix as long as the total number of elements remains unchanged	"
CØLMAX, CØLMIN	Searches an input matrix to obtain the maximum or minimum values within each column	"
SYMREM, SYMREP	Allows the SINDA user to operate on a simple row/column of a half symmetric matrix	A.6-14
SYMDAD	Adds the elements of a vector array to the corresponding elements of the main diagonal of a half symmetric matrix	"
SYMIV	Obtains the inverse of a half symmetric matrix	"
PØLMLT	Multiplies a given number of nth order polynomial coefficients by a similar number of mth order polynomial coefficients	A.6-15
PØLVAL	Evaluates the polynomial for the input complex number X + iY, given a set of polynomial coefficients	"
PLYEVL	Evaluates each polynomial for each X value, given a matrix with nth order polynomial coefficients and a column matrix of X values	"
PØLSØV	Calculates the complex roots, given a set of polynomial coefficients as the first row in a matrix	"
JACØBI	Determines the eigenvalues and eigenvector associated with an input matrix [A]	A.6-16

Store and Recall

CALL	Retrieves matrices on magnetic tape	A.6-17
FILE	Stores matrices on magnetic tape	"
ENDMØP	Used in conjunction with subroutines CALL and FILE. Causes all matrices from the logical 12 tape to be updated onto the logical 13 tape	"
LSTAPE	Will output the name, problem number and size of every matrix stored on tape on logical 13	"

Applications

MØDES	Solves a particular matrix dynamic vibration equation	A.6-18
MASS	Generates an inertia matrix of a dynamic vibration system described in terms of deflections and rotations	A.6-19
STIFF	Generates a stiffness matrix for a dynamic vibration system described in terms of deflections and rotations	A.6-20

SUBROUTINE NAMES: ZERØ or ØNES

PURPOSE: These subroutines generate a matrix [Z] such that every element is zero or one respectively.

RESTRICTIONS: The matrix to be generated must contain exactly enough space in addition to having the integer number of rows and columns as the first two data values. The NR and NC arguments are the integer number of rows and columns respectively.

CALLING SEQUENCE: ZERØ(NR,NC,Z(IC))

or ØNES(NR,NC,Z(IC))

SUBROUTINE NAMES: UNITY or SIGMA

PURPOSE: These are square matrix generation subroutines. UNITY generates a square matrix such that the main diagonal elements are one and all other elements are zero. SIGMA generates a square matrix such that all elements on and below the main diagonal are one and the remaining elements are zero.

RESTRICTIONS: The matrix [Z] to be generated must contain exactly enough space in addition to having the integer number of rows and columns as the first two data values. The integer number of rows and columns are equal and must be input as the argument N.

CALLING SEQUENCE: UNITY(N,Z(IC))

or SIGMA(N,Z(IC))

SUBROUTINE NAMES: GENALP or GENCØL

PURPOSE: These are special matrix generation subroutines. GENALP will generate a matrix such that every element is equal to a constant C. GENCØL will generate a column matrix such that the first element is equal to X1 and the last element is equal to X2. The intermediate elements receive equally incremented values such that a linear relationship is established between row number and element value.

RESTRICTIONS: The NR and NC arguments refer to the integer number of rows and columns respectively. X1, X2 and C must be floating point values. The generated matrices must contain exactly enough space in addition to having the integer number of rows and columns as the first two data values.

CALLING SEQUENCE: GENALP(NR,NC,C,Z(IC))

or GENCØL(X1,X2,NR,Z(IC))

SUBROUTINE NAMES: FULSYM or SYMFUL

These subroutines allow the SINDA user to form a half symmetric matrix from a full square matrix or form a full square matrix from a half symmetric matrix, respectively. The arguments must address the matrix array integer count set by the preprocessor, the array lengths must be exact.

RESTRICTIONS:

The half symmetric matrix must be formatted as shown on page A.8-8 and the full square matrix must be formatted as described on page A.6-1 of this document.

CALLING SEQUENCE: FULSYM(FM(IC),SM(IC))

or SYMFUL(SM(IC),FM(IC))

Where FM is the full matrix and SM is the symmetric matrix.

SUBROUTINE NAME: SYMFRC

PURPOSE:

This subroutine may be used to force symmetry upon a square matrix. The main diagonal elements are untouched and all others are treated as follows:

$$x = (a_{ij} + a_{ji})/2.0;\ a_{ij} = x;\ a_{ji} = x$$

RESTRICTIONS:

The addressed matrix must be square and formatted as described on page

CALLING SEQUENCE: SYMFRC(A(IC))

SUBROUTINE NAMES: DIAG or UNDIAG or DIAGAD

PURPOSE:

Given a 1*N or N*1 matrix [V], subroutine DIAG forms a full square N*N matrix [Z]. The [V] values are placed sequentially on the main diagonal of [Z] and all off diagonal elements are set to zero. Subroutine UNDIAG forms a 1*N matrix [V] from the diagonal elements of an N*N matrix [Z]. Subroutine DIAGAD adds the elements of a 1*N matrix [V] to the diagonal elements of an N*N matrix [Z].

RESTRICTIONS:

Both matrices must have exactly enough space and contain their integer number of rows and columns as the first two data values.

CALLING SEQUENCE: DIAG(V(IC),Z(IC))

or UNDIAG(Z(IC),V(IC))

or DIAGAD(V(IC),Z(IC))

SUBROUTINE NAMES: ELEADD or ELESUB

PURPOSE: These subroutines add or subtract the corresponding elements of two matrices respectively.

$$[Z]^{m*n} = [A]^{m*n} \pm [B]^{m*n}, \quad z_{ij} = a_{ij} \pm b_{ij}$$

RESTRICTIONS: All matrices must be of identical size and have the integer number of rows and columns as the first two data values. The [Z] matrix may be overlayed into the [A] or [B] matrix.

CALLING SEQUENCE: ELEADD(A(IC),B(IC),Z(IC))
or ELESUB(A(IC),B(IC),Z(IC))

SUBROUTINES NAMES: ELEMUL or ELEDIV

PURPOSE: These subroutines multiply or divide the corresponding elements of two matrices respectively.

$$[Z]^{m*n} = [A]^{m*n} \,*/\, [B]^{m*n}, \quad a_{ij} = a_{ij} \,*/\, b_{ij}$$

RESTRICTIONS: All matrices must be of identical size and have the integer number of rows and columns as the first two data values. The [Z] matrix may be overlayed into the [A] or [B] matrix.

CALLING SEQUENCE: ELEMUL(A(IC),B(IC),Z(IC))
or ELEDIV(A(IC),B(IC),Z(IC))

SUBROUTINE NAME: ELEINV

PURPOSE: This subroutine obtains the reciprocal of each element of the A matrix and places it in the corrresponding element location of the [Z] matrix.

$$z_{ij} = 1.0/a_{ij}$$

RESTRICTIONS: The matrices must be of identical size and have the integer number of rows and columns as the first two data values. The [Z] matrix may be overlayed into the [A] matrix.

CALLING SEQUENCE: ELEINV(A(IC),Z(IC))

SUBROUTINE NAMES: EFSIN or EFASN

PURPOSE: These subroutines perform elementary functions on all of the [A] matrix elements as follows:

$$z_{ij} = \sin(a_{ij}) \quad \text{or} \quad z_{ij} = \arcsin(a_{ij})$$

RESTRICTIONS: The matrices must be identical in size and have the integer number of rows and columns as the first two data values. The [Z] matrix may be overlayed into the [A] matrix.

CALLING SEQUENCE: EFSIN(A(IC),Z(IC))

or EFASN(A(IC),Z(IC))

SUBROUTINE NAMES: EFCØS or EFACS

PURPOSE: These subroutines perform elementary functions on all of the [A] matrix elements as follows:

$$z_{ij} = \cos(a_{ij}) \quad \text{or} \quad a_{ij} = \arccos(a_{ij})$$

RESTRICTIONS: The matrices must be identical in size and have the integer number of rows and columns as the first two data values. The [Z] matrix may be overlayed into the [A] matrix.

CALLING SEQUENCE: EFCØS(A(IC),Z(IC))

or EFACS(A(IC),Z(IC))

SUBROUTINE NAMES: EFTAN or EFATN

PURPOSE: These subroutines perform elementary function on all of the [A] matrix elements as follows:

$$z_{ij} = \text{tangent}(a_{ij}) \quad \text{or} \quad z_{ij} = \text{arctangent}(a_{ij})$$

RESTRICTIONS: The matrices must be of identical size and have the integer number of rows and columns as the first two data values. The [Z] matrix may be overlayed into the [A] matrix.

CALLING SEQUENCE: EFTAN(A(IC),Z(IC))

or EFATN(A(IC),Z(IC))

ELEMENTAL OPERATIONS

SUBROUTINE NAMES: EFABS or EFLØG or EFSQR

PURPOSE: These subroutines perform elementary functions on all of the [A] matrix elements as follows respectively:

$$z_{ij} = |a_{ij}| \quad \text{or} \quad z_{ij} = \log_e(a_{ij}) \quad \text{or} \quad z_{ij} = \sqrt{a_{ij}}$$

RESTRICTIONS: The matrices must be identical in size and have the integer number of rows and columns as the first two data values. All in the [A] matrix must be positive for EFLØG or EFSQR.

CALLING SEQUENCE:

```
EFABS(A(IC),Z(IC))
EFLØG(A(IC),Z(IC))
EFSQR(A(IC),Z(IC))
```

SUBROUTINE NAMES: EFEXP or EFPØW

PURPOSE: These subroutines perform elementary functions on all of the [A] matrix elements as follows:

$$z_{ij} = e^{a_{ij}} \quad \text{or} \quad z_{ij} = a_{ij}^{\alpha}$$

RESTRICTIONS: The matrices must be identical in size and have the integer number of rows and columns as the first two data values. The [Z] matrix may be overlayed into the [A] matrix. The exponent α may be an integer or floating point number. However, if any elements in [A] are negative then α must be an integer.

CALLING SEQUENCE:

```
EFEXP(A(IC),Z(IC))
```

or

```
EFPØW(A(IC),α,Z(IC))
```

ELEMENTAL OPERATIONS

SUBROUTINE NAMES: ADDALP or ALPHAA

PURPOSE:

To add a constant to or multiply a constant times every element in a matrix.

$$z_{ij} = C + a_{ij} \quad \text{or} \quad z_{ij} = C*a_{ij}$$

RESTRICTIONS:

The matrices must have exactly enough space and contain the integer number of rows and columns as the first two data values. C and all elements must be floating point numbers. The [Z] matrix may be overlayed into the [A] matrix.

CALLING SEQUENCE: ADDALP(C,A(IC,Z(IC))

or ALPHAA(C,A(IC),Z(IC))

SUBROUTINE NAMES: MATRIX or SCALAR or MATADD

PURPOSE: The subroutine MATRIX allows a constant to replace a specific matrix element, subroutine SCALAR allows a specific matrix element to be placed into a constant location, and subroutine MATADD adds a constant to a specific matrix element. The integers I and J designate the row and column position of the specific element.

$$z_{ij} = C \quad \text{or} \quad C = z_{ij} \quad \text{or} \quad z_{ij} = z_{ij} + C$$

RESTRICTIONS: The matrix must have the integer number of rows and columns as the first two data values. Checks are made to insure that the identified element is within the matrix boundaries.

CALLING SEQUENCE: MATRIX(C,I,J,Z(IC))

or SCALAR(Z(IC),I,J,C)

or MATADD(C,I,J,Z(IC))

SUBROUTINE NAME: INVRSE

PURPOSE: To invert a square matrix.

$$\text{given } [A]_{n*n} \quad , \quad [Z]_{n*n} = [A]^{-1}_{n*n}$$

RESTRICTIONS: The matrices must be square, identical in size and contain the integer number of rows and columns as the first two data values. The output matrix [Z] may be overlayed into the [A] matrix.

CALLING SEQUENCE: INVRSE(A(IC),Z(IC))

NOTE: This subroutine requires n dynamic storage allocations.

SUBROUTINE NAME: MULT

PURPOSE: To multiply two conformable matrices together.

$$[Z]_{m*n} = [A]_{m*p} \; [B]_{p*n} \quad , \quad z_{ij} = a_{ik} * b_{kj}$$

RESTRICTIONS: The matrices must have exactly enough space and contain their integer number of rows and columns as the first two data values. If [A] and [B] are square, [Z] may be overlayed into either of them.

CALLING SEQUENCE: MULT(A(IC),B(IC),Z(IC))

NOTE: This subroutine requires n*m dynamic storage locations.

SUBROUTINE NAME: TRANS

PURPOSE:

Given a matrix $[A]_{m*n}$ form its transpose as $[Z]_{n*m}$

RESTRICTIONS: Both matrices must have exactly enough space and contain their integer number of rows and columns as the first two data values. The output matrix [Z] may be overlayed into the [A] matrix.

CALLING SEQUENCE: TRANS(A(IC),Z(IC))

NOTE: This subroutine requires n*m dynamic storage locations.

MATRIX OPERATIONS AND SOLUTIONS

SUBROUTINE NAME: AABB

PURPOSE:

To sum two scaled matrices:

$$\underset{[Z]}{m*n} = \underset{C1[A]}{m*n} + \underset{C2[B]}{m*n}, \quad z_{ij} = C1*a_{ij} + C2*b_{ij}$$

RESTRICTIONS:

All matrices must be of identical size, contain exactly enough space and contain the integer number of rows and columns as the first two data values. The output matrix [Z] may be overlayed into either of the input matrices.

CALLING SEQUENCE: AABB(C1,A(IC),C2,B(IC),Z(IC))

SUBROUTINE NAMES: BTAB or BABT

PURPOSE:

To perform the following matrix operations, respectively:

$$\underset{[Z]}{n*m} = \underset{[B]}{n*m}^{t} \; \underset{[A]}{m*m} \; \underset{[B]}{m*m}$$

or

$$\underset{[Z]}{m*m} = \underset{[B]}{m*n} \; \underset{[A]}{n*n} \; \underset{[B]}{n*m}^{t}$$

RESTRICTIONS:

The matrices must be conformable, contain exactly enough space and contain the integer number of rows and columns as the first two data values. Subroutines MULT and TRANS are called on.

CALLING SEQUENCE: BTAB(A(IC),B(IC),Z(IC))

or BABT(A(IC),B(IC),Z(IC))

NOTE: Due to subroutines MULT and TRANS this subroutine temporarily requires 2*m*n+6 dynamic storage locations.

MATRIX OPERATIONS AND SOLUTIONS

SUBROUTINE NAMES: DISAS or ASSMBL

PURPOSE:

These subroutines allow a user to operate on matrices in a partitioned manner by disassembling a submatrix [Z] from a parent matrix [A] or assembling a submatrix [Z] into a parent matrix [A].

RESTRICTIONS:

The I and J arguments are integers which identify (by row and column number respectively) the upper left hand corner position of the submatrix within the parent matrix. All matrices must have exactly enough space and contain the integer number of rows and columns as the first two data values. The NR and NC arguments are the integer number of rows and columns respectively of the disassembled submatrix. If the submatrix exceeds the bounds of the parent matrix an appropriate error message is written and the program terminated.

CALLING SEQUENCE: DISAS(A(IC),I,J,NR,NC,Z(IC))

or ASSMBL(Z(IC),I,J,A(IC))

SUBROUTINE NAMES: CØLMLT or RØWMLT

PURPOSE:

To multiply each element in a column or row of matrix [A] by its corresponding element from the matrix [V] which is conceptually a diagonal matrix but stored as a vector; i.e., 1*N or N*1 matrix. The matrix [Z] is the product.

RESTRICTIONS:

The matrices must have exactly enough space and contain the integer number of rows and columns as the first two data values. The matrices being multiplied must be conformable.

CALLING SEQUENCE: CØLMLT(A(IC),V(IC),Z(IC))

or RØWMLT(V(IC),A(IC),Z(IC))

MATRIX OPERATIONS AND SOLUTIONS

SUBROUTINE NAMES: SHIFT or REFLCT*

PURPOSE: These subroutines may be used to move an entire matrix from one location to another SHIFT moves the matrix exactly as is and REFLCT moves it and reverses the order of the elements within each column. The last element in each column becomes the first and the first becomes the last, etc.

RESTRICTIONS: The matrices must be of identical size and the integer number of rows and columns must be the first two data values. The [Z] matrix may be overlayed into the [A] matrix.

CALLING SEQUENCE: SHIFT(A(IC),Z(IC))

or REFLCT(A(IC),Z(IC))

*REFLCT uses three dynamic storage locations plus an additional one for each row.

SUBROUTINE NAME: SHUFL

PURPOSE: This subroutine allows the user to reorder the size of a matrix as long as the total number of elements remains unchanged. The row order input matrix [A] is transposed to achieve column order and then reformed as a vector by sequencing the columns in ascending order. This vector is then reformed into a column order matrix by taking a column at a time sequentially from the vector. The newly formed column matrix is then transposed and output as the row order matrix [Z].

RESTRICTIONS: The matrices must be identical in size and have their respective integer number of rows and columns as the first two data values. The number of rows times columns for [A] must equal the number of rows times columns of [Z].

CALLING SEQUENCE: SHUFL(A(IC),Z(IC))

SUBROUTINE NAMES: CØLMAX or CØLMIN

PURPOSE: These subroutines search an input matrix to obtain the maximum or minimum values within each column respectively. These values are output as a single row matrix [A] having as many columns as the input matrix [A].

RESTRICTIONS: Each matrix must have its integer number of rows and columns as the first two data values.

CALLING SEQUENCE: CØLMAX(A(IC),Z(IC))

or CØLMIN(A(IC),Z(IC))

MATRIX OPERATIONS AND SOLUTIONS

SUBROUTINE NAMES: SYMREM or SYMREP

PURPOSE:

These subroutines allow the SINDA user to operate on a single row/column of a half symmetric matrix. SYMREM will remove a particular row/column from the half symmetric matrix and place it into an array of the exact length to hold it. SYMREP will take an array and replace it into a specific row/column of the half symmetric matrix.

RESTRICTIONS:

The half symmetric matrix must be formatted as shown on page A.8-8. The integer K must designate the row/column to be operated on. If K is an integer zero the main diagonal will be removed or replaced.

CALLING SEQUENCE: SYMREM(K,SM(IC),A(IC))

or SYMREP(K,A(IC),SM(IC))

SUBROUTINE NAME: SYMDAD

PURPOSE:

This subroutine will add the elements of a vector array to the corresponding elements of the main diagonal of a half symmetric matrix. If any sum of the elements is less than zero they are set to zero.

RESTRICTIONS:

The half symetric matrix must be formatted as shown on page A.8-8. The vector array must be input as a positive array and be the same length as the matrix order.

CALLING SEQUENCE: SYMDAD(VA(IC),SM(IC))

SUBROUTINE NAME: SYMINV

PURPOSE:

This subroutine obtains the inverse of a half symmetric matrix which is also symmetric and returns it in the same area as the input matrix. This subroutine is called internally by subroutines SCRPFA, IRRADI and SLRADI.

RESTRICTIONS:

This subroutine contains no error checks, exercise extreme caution when using it.

CALLING SEQUENCE: SYMINU(A(DV),N)

where A(DV) addresses the 1,1 element and N is the matrix order.

SUBROUTINE NAME: PØLMLT

PURPOSE: This subroutine performs the multiplication of a given number of n^{th} order polynomial coefficients by a similar number of m^{th} order polynomial coefficients. The polynomials must be input as matrices with the number of rows equal and each row receives the following operation.

$$(c_1,c_2,c_3,\ldots,c_k) = (a_1,a_2,\ldots,a_n) * (b_1,b_2,\ldots,b_m), k=m+n-1$$

RESTRICTIONS: The matrices must have exactly enough space and contain their integer number of rows and columns as the first two data values.

CALLING SEQUENCE: PØLMLT(A(IC),B(IC),C(IC))

SUBROUTINE NAME: PØLVAL

PURPOSE: Given a set of polynomial coefficients as the first row of matrix [A], this subroutine evaluates the polynomial for the input complex number X+iY. The answer is returned as U+iV.

RESTRICTIONS: [A] may be m*n but only the first row is evaluated.

CALLING SEQUENCE: PØLVAL(A(IC),X,Y,U,V)

SUBROUTINE NAME: PLYEVL

PURPOSE: Given a matrix [A] containing an arbitrary number NRA of the n^{th} order polynomial coefficients and a column matrix [X] containing an arbitrary number of NRX of x values, this subroutine evaluates each polynomial for X value. The answers are output as a matrix [Z] of size NRX*NRA. Each set of polynomial coefficients in [A] is a row in ascending order. An x value evaluated for the polynomial creates a row in [Z] where the column number agrees with the polynomial row number.

RESTRICTIONS: The matrices must have exactly enough space and contain their integer number of rows and columns as the first two data values.

CALLING SEQUENCE: PLYEVL(A(IC),X(IC),Z(IC))

SUBROUTINE NAME: PØLSØV

PURPOSE: Given a set of polynomial coefficients as the first row in matrix [A], size (m,n+1), this subroutine calculates the complex roots which are returned as matrix [Z], size (n,2). Column 1 contains the real part and column 2 imaginary part of the roots.

RESTRICTIONS: This subroutine presently is limited to n = 20. It internally calls on RTPØLY and utilizes some double precision.

CALLING SEQUENCE: PØLSØV(A(IC),Z(IC))

SUBROUTINE NAME: JACØBI

PURPOSE:

This subroutine will find the eigenvalues [E] and eigenvector matrix [Z] associated with an input matrix [A].

n*n	n*n		n*n	n*n
[A]	[Z]	=	[Z]	[E]

RESTRICTIONS:

The matrices must have exactly enough space and contain their integer number of rows and columns as the first two data values. Note that matrix [E] is a diagonal matrix but is stated as a vector.

CALLING SEQUENCE: JACØBI(A(IC),E(IC),Z(IC))

NOTE: This subroutine requires 2*n*n+6 dynamic storage locations.

STORE AND RECALL

MATRIX DATA STORAGE AND RETRIEVAL

The ability to store and retrieve matrices from tape is easily achieved through the use of the FILE and CALL subroutines. Matrices are identified by an alphanumeric name, integer problem number and the core address of or for the matrix. The CALL subroutine searches the matrix storage tape on logical 13 and brings the desired matrix into core. The FILE subroutine writes a matrix onto the logical 12 tape. Subroutine ENDMØP causes all matrices from the logical 12 tape to be updated onto the logical 13 tape. in case of duplicate matrices, the one from logical 12 replaces the one on logical 13. A matrix which has been filed cannot be called until an ENDMØP operation has been performed. To create a new tape the user merely sets control constant NØCØPY nonzero and has a scratch tape mounted on logical 13. The user should check the section on control cards and deck setup to determine control card requirements. (Appendix E)

SUBROUTINE NAMES: CALL or FILE

PURPOSE:

To allow the user to retrieve or store matrices on magnetic tape, see above The H argument must be a six character alphanumeric word and N must be an integer number, both of which are used to identify the matrix.

RESTRICTIONS:

See above. The matrix must have exactly enough space and contain the integer number of rows and columns as the first two data values.

CALLING SEQUENCE: CALL(H,N,A(IC))
or FILE(A(IC),H,N)

SUBROUTINE NAMES: ENDMØP or LSTAPE

PURPOSE:

Subroutine ENDMØP should be used in conjunction with subroutines CALL and FILE, see above. It causes matrices which have been filed by FILE on logical 12 to be updated onto logical 13. A call to subroutine LSTAPE will cause the output of the name, problem number and size of every matrix stored on tape on logical 13.

RESTRICTIONS: See above.

CALLING SEQUENCE: ENDMØP
or LSTAPE

APPLICATION -- DYNAMIC VIBRATION

SUBROUTINE NAME: MØDES

PURPOSE:

This subroutine solves the following dynamic vibration equation

$$\underset{[A]}{n*n}\ \underset{[Z]}{n*n} = \underset{[B]}{n*n}\ \underset{[Z]}{n*n}\ \underset{\left[\frac{1}{W^2}\right]}{n*n}$$

where [A] is the input inertia matrix associated with the kinetic energy and [B] is the input stiffness matrix associated with the strain energy. [Z] is the output eigenvector matrix associated with the frequencies of vibration W_i which are output in radians/sec as [R] and in cycles/sec as [C], both [R] and [C] are n*n diagonal matrices but stored as vectors.

RESTRICTIONS:

The matrices must have exactly enough space and contain their integer number of rows and columns as the first two data values. Subroutine JACØBI is called on.

CALLING SEQUENCE: MØDES(A(IC),B(IC),Z(IC),R(IC),C(IC))

NOTE: This subroutine requires 3*n*n+9 dynamic storage locations.

APPLICATION -- DYNAMIC VIBRATION

SUBROUTINE NAME: MASS

PURPOSE:

If a dynamic vibration problem is referred to a set of coordinates consisting of the deflections, ζ_i, and the rotations, θ_i, at N collocation points along the beam under consideration, then this subroutine generates the 2N by 2N inertia matrix [A] which appears in the following expression for kinetic energy:

$$T = \frac{1}{2} \left\{ \dot{\zeta}_1 \ldots \dot{\zeta}_n \dot{\theta}_1 \ldots \dot{\theta}_n \right\} [A] \begin{bmatrix} \dot{\zeta}_1 \\ \vdots \\ \dot{\zeta}_n \\ \dot{\theta}_1 \\ \vdots \\ \dot{\theta}_n \end{bmatrix}$$

RESTRICTIONS:

The mass and inertia data input to this subroutine are to be supplied as piecewise continuous slices; however, these arrays may be of arbitrary size and different in length from each other. The number of collocation points, N, which determines the ultimate size, 2N by 2N, of the output inertia matrix, is also chosen arbitrarily.

CALLING SEQUENCE: MASS(X(IC),DMPL(IC),RIPL(IC),CM(IC),A(IC))

where X is the matrix (N X 1) of collocation points referred to an arbitrary origin.

DMPL is the matrix(NDM X 4) of distributed mass per unit length slices, where
- Col 1 is the location of the rear of a slice.
- Col 2 is the location of the front of a slice.
- Col 3 is the mass value at the rear of the slice.
- Col 4 is the mass value at the front of the slice.

RIPL is the matrix (NRI X 4) of distributed rotary inertia per unit length slices. The columns here are similar to DMPL.

CM is the matrix (NCM X 4) of concentrated mass items, where
- Col 1 is the attach point location for each item.
- Col 2 is the mass at this location.
- Col 3 is the location of its center of gravity.
- Col 4 is the moment of inertia about the C. of G.

A is the output (2N X 2N) inertia matrix.

NOTE: Having application to DMPL, RIPL and CM, it is noted that the location of the values may not go beyond the limits of the collocation points in either direction.

SUBROUTINE NAME: STIFF

PURPOSE:

If a dynamic vibration problem is referred to a set of coordinates consisting of the deflections, ζ_i, and the rotations, θ_i, at N collocation points along the beam under consideration, then this subroutine generates the 2N by 2N stiffness matrix [K] which appears in the following expression for the strain energy:

$$U = \frac{1}{2} \{\zeta_1 \cdots \zeta_n \theta_1 \cdots \theta_n\} [K] \begin{bmatrix} \zeta_1 \\ \cdot \\ \cdot \\ \cdot \\ \zeta_n \\ \theta_1 \\ \cdot \\ \cdot \\ \cdot \\ \theta_n \end{bmatrix}$$

RESTRICTIONS:

The stiffness and shear data input to this subroutine are to be supplied as piecewise continuous slices; however, these arrays may be of arbitrary size and different in length from each other. The number of collocation points, N, which determine the ultimate size, 2N by 2N, of the output stiffness matrix, is also chosen arbitrarily.

CALLING SEQUENCE: STIFF(X(IC),EI(IC),GA(IC),K(IC))

where X is the matrix (N X 1) of collocation points referred to an arbitrary origin.

EI is the matrix (NEI X 4) of bending stiffness slices, where
Col 1 is the location of the rear of a slice.
Col 2 is the location of the front of a slice.
Col 3 is the stiffness value at the rear of a slice.
Col 4 is the stiffness value at the front of a slice.

GA is the matrix (NGA X 4) of shear stiffness slices, where the columns here are similar to those for the EI distribution.

K is the output stiffness matrix size 2N by 2N.

NOTE: Having application to EI and GA, it is noted that the location of the values may not go beyond the limits of the collocation points in either direction.

A.7 OUTPUT SUBROUTINES

Data Input and Temperature Printout

GPRINT QFPRNT	Causes the printout of all conductor values or heat flow rates through conductors	A.7-3
TPRINT CPRINT QIPRNT QNPRNT	Causes the printout of all nodal temperatures, all capacitance values, all impressed heating rates or all net heat rates for the nodal network under consideration	"

Numerical Differencing Characteristics Printout

STNDRD	Causes a line of output to be printed giving present time, last time step used, most recent CSGMIN, maximum diffusion change calculated over the last time step and maximum relaxation calculated over the last iteration	A.7-4
PRNTMP	Calls on STNDRD and also lists temperature of every node in the network according to relative node number	"

Floating Point

PRINT PRINTL	Allows individual floating point numbers to be printed for reference temperature, capacitance, etc.	"

Array Printout

PRINTA	Allows the user to printout an array of values five to the line	A.7-5
PRNTMA	Allows the user to print up to 10 arrays in a column format	"
PUNCHA	Enables a user to punch out an array of data values in any desired format	A.7-6
PNCHMA	Similar to PUNCHA but up to 10 equal length arrays of data values may be punched	"

Plot Package

PLØTX1	Call upon a large package of undocumented subroutines specifically for the SC-4060	A.7-7
PLØTX2		"
PLØTL1		"
PLØTL2		"
PLØTX3		A.7-8
PLØTX4		"
SC-4060	Plot Symbol Dictionary	A.7-9

Read and Rewind

READ, WRITE	Enables the user to read and write arrays of data as binary information on magnetic tape	A.7-10 "
EØF, REWIND	Enables the user to write end of file marks on magnetic tape and to rewind them	" "
LIST	Prints the elements of a matrix and identifies each by its row and column number	A.7-11
PUNCH	Punches out a matrix, size n*n, one column at a time in any desired format	"
SYMLST	Prints out and identifies the element values of a half symmetric matrix	"
PNTABL	Provides output information for users of subroutine ABLATS	A.7-12

SUBROUTINE NAMES: GPRINT or QFPRNT

PURPOSE:

These subroutines cause the printout of all conductor values or heat flow rates through conductors. All values are printed out versus the actual conductor numbers on which they occur. When using either of these subroutines the user must allocate one dynamic storage location for each conductor in the system. The locations are permanently retained for storage of the actual conductor numbers and is common to all subroutines* requiring them. In addition, subroutine QFPRNT requires one extra dynamic storage location per conductor for temporary storage of heat flow rates through the conductors.

RESTRICTIONS:

These subroutines require no arguments and are generally called from the EXECUTION or OUTPUT CALLS block; do not call them from VARIABLES 1. Nonlinear conductors are evaluated prior to calculation and/or printing of requested values.

CALLING SEQUENCE:

GPRINT or QFPRNT

*For example, the actual conductor numbers stored by GPRINT are available to CSGDMP and RCDUMP, thereby conserving dynamic storage.

SUBROUTINE NAMES: TPRINT or CPRINT or QIPRNT or QNPRNT

PURPOSE:

These subroutines cause the printout of all nodal temperatures, all capacitance values, all impressed heating rates or all net heating rates for the nodal network under consideration. All values are printed out versus the actual node numbers on which they occur. When using any of these subroutines the user must allocate one dynamic storage location for each node in the system. The locations are permanently retained for storage of the actual node numbers and is common to all subroutines* requiring them. It should be noted that TPRINT call on STNDRD (page A.7-4).

RESTRICTIONS:

These subroutines require no arguments and are generally called from the EXECUTION or OUTPUT CALLS block; do not call them from VARIABLES 1. Nonlinear network elements are evaluated prior to calculation and/or printing of requested values.

CALLING SEQUENCE:

TPRINT or CPRINT or QIPRNT or QNPRNT

*For example, the actual node numbers stored by TPRINT are available to CSGDMP and RCDUMP, thereby conserving dynamic storage.

NUMERICAL DIFFERENCING CHARACTERISTICS FLOATING POINT PRINTOUT

SUBROUTINE NAMES: STNDRD or PRNTMP

PURPOSE:

Subroutine STNDRD causes a line of output to be printed giving the present time, the last time step used, the most recent CSGMIN value, the maximum diffusion temperature change calculated over the last time step and the maximum relaxation change calculated over the last iteration. ANN refers to the actual node number on which something occurred. The line of output looks as follows:

```
*   *   *   *
TIME____DTIMEU____CSGMIN(ANN)_____TEMPCC(ANN)____RELXCC(ANN)____
```

Subroutine PRNTMP internally calls on STNDRD and also lists the temperature of every node in the network according to relative node number. The relative node number - actual node number dictionary printed out with the input data should be consulted to determine temperature locations on the thermal network model.

RESTRICTIONS:

No arguments are required or allowed. These subroutines should be used with network problems only.

CALLING SEQUENCE:

```
STNDRD
or PRNTMP
```

SUBROUTINE NAMES: PRINT or PRINTL

PURPOSE:

These subroutines allow individual floating point numbers to be printed. The arguments may reference temperature, capacitance, source locations, conductors, constants or unique array locations. In addition, subroutine PRINTL allows each value to be preceded or labeled by a six character alphanumeric word. The number of arguments is variable but the "label" array used for PRINTL should contain a label for each argument.

RESTRICTIONS:

These subroutines do not call on STNDRD. The user may call on it if he desires time control information. Any control constant may be addressed in order to see what its value is, integers must first be floated.

CALLING SEQUENCE:

```
PRINT(T,C,Q,G,K,...,A+)
or PRINTL(LA(DV),T,C,Q,G,K,...,A+)
```

ARRAY PRINTOUT

SUBROUTINE NAME: PRINTA

PURPOSE:

This subroutine allows the user to print out an array of values, five to the line. The integer array length N and the first data value location must be specified. Each value receives an indexed label, the user must supply a six character alphanumeric word L to be used as a common label and an integer value M to begin the index count.

RESTRICTIONS:

The array values to be printed must be floating point numbers.

CALLING SEQUENCE:

PRINTA(L,A(DV),N,M)

If the label was the word TEMP, N was 3 and M was 6 the line of output would look as follows:

TEMP (6)value TEMP (7)value TEMP (8)value

SUBROUTINE NAME: PRNTMA or PRNTMI*

PURPOSE:

This subroutine allows the user to print out up to 10 arrays in a column format. The individual elements are not labeled but each column receives a two line heading of 12 alphanumeric characters each. The two line heading must be supplied as a single array of four words, six characters each. The user must supply the starting location of each label array and value array. The number of values in each value array must agree and be supplied as the integer N. The value arrays must contain floating point numbers.

RESTRICTIONS:

Labels must be alphanumeric while values must be floating point. All floating point value arrays must contain the same number of values.

CALLING SEQUENCE:

PRNTMA(N,LA1(DV),VA1(DV),LA2(DV),VA2(DV),...)
PRNTMI(N,LA1(DV),VA1(DV),LA2(DV),VA2(DV),...)

* VA1 must address array of integers (1st column)

ARRAY PRINTOUT

SUBROUTINE NAME: PUNCHA

PURPOSE:

This subroutine enables a user to punch out an array of data values in any desired format. The F argument must reference a FØRTRAN FØRMAT which has been input as an array, including the outer parenthesis but deleting the word FØRMAT.* The second argument must address the first data value of the array of sequential values. The third argument, N, must be the integer number of data values in the array. The output is written onto logical tape 15, the user must provide the necessary control cards and processing information for the operator.

RESTRICTIONS:

The user should check Appendix E for the appropriate control card requirements. Punched output is written on logical tape 15, operator processing instructions should be supplied.

CALLING SEQUENCE: PUNCHA (F(DV),A(DV),N)

SUBROUTINE NAME: PNCHMA

PURPOSE:

This subroutine is similar to PUNCHA, but up to 10 equal length arrays of data values may be punched. Again the first argument must reference a FØRTRAN FØRMAT which has been input as an array, including the outer parenthesis, but deleting the word FØRMAT. The integer number of data values in an array must be supplied as the second argument N. The array starting locations then follow as arguments three up to twelve. The first value in each array is punched, then the second, etc.

RESTRICTIONS:

The user should check Appendix E for the appropriate control card requirements. Punched output is written on logical tape 15, operator processing instructions should be supplied.

CALLING SEQUENCE: PNCHMA(F(DV),N,A1(DV),A2(DV),...)

* For example, if F(DV) were A5+1, A5 could be input as follows:

```
(Col)  7    12
            5
       BCD  4(12X,5(F9.3,1H,),F9.3)
       END
```

SUBROUTINE NAMES: PLØTX1 or PLØTX2 or PLØTL1 or PLØTL2

PURPOSE:

These FØRTRAN V coded quick plot subroutines call upon a large package of undocumented subroutines specifically for the SC 4060. They will produce up to four graphs per frame and several variables may be plotted per graph. A suitable grid will be drawn with certain lines emphasized. The grid lines will have reasonable numerical indicia and centered title will be printed for both axes and at the top of the graph.

PLØTX1 and PLØTL1 will compute the minimum and maximum values of the stored X and Y arrays to be plotted and calls upon PLØTX2 or PLØTL2 which use the values as grid limits for the graph. The user may set the grid limits by calling PLØTX2 and PLØTL2 directly. The X, Y and top titles (XT, YT and TT respectively) must consist of 9 alphanumeric words of six characters each.

RESTRICTIONS:

The user should consult Appendix E, Control Cards and Deck Setup to check tape designation requirements. The X and Y values must be floating point numbers. The user must call subroutine PLTND after all his plotting is done. No limit may be zero for log plots.

CALLING SEQUENCE:

```
      PLØTX1(N,IS,TX(DV),TY(DV),TT(DV),NP,AX(DV),AY(DV))
   or
      PLØTX2(N,XL,XR,YB,YT,IS,TX(DV),TY(DV),TT(DV),NP,AX(DV),AY(DV))

      PLØTL1(N,IS,TX(DV),TY(DV),TT(DV),NP,AX(DV),AY(DV),LM)
   or
      PLØTL2(N,XL,XR,YB,YT,IS,TX(DV),TY(DV),TT(DV),NP,AX(DV),AY(DV),LM)
```

Where N is the integer number of graphs per frame (1, 2, 3 or 4), if zero, the grid from the previous plot call is used.
IS is the integer identifying the plotting symbol (1-144)
TX is the address of the X title
TY is the address of the Y title
TT is the address of the top title
NP is the integer number of XY values or points to be plotted, if negative the points will be connected by straight lines.
AX is the address of the X array
AY is the address of the Y array
XL is the floating point X axis left limit
XR is the floating point X axis right limit
YB is the floating point Y axis bottom limit
YT is the floating point Y axis top limit
LM is an integer identifying the log plotting mode; if less than zero plot log X versus linear Y, if equal to zero plot log X versus log Y, if greater than zero plot linear X versus log Y

SUBROUTINE NAMES: PLØTX3 or PLØTX4

PURPOSE:

These subroutines are similar to PLØTX1 and PLØTX2 but have 6 additional arguments which allow the user to modify the grid as desired.

RESTRICTIONS:

See PLØTX1 and PLØTX2.

CALLING SEQUENCE:

```
PLØTX3(N,IS,TX(DV),TY(DV),TT(DV),NP,AX(DV),AY(DV),DX,DY,L,M,I,J)
```

or

```
PLØTX4(N,XL,XR,YB,YT,IS,TX(DV),TY(DV),TT(DV),NP,AX(DV),AY(DV),DX,
DY,L,M,I,J)
```

where the arguments are identical to PLØTX1 and PLØTX2 except for

DX,DY	these floating point values are used for spacing the grid lines which are centered on the zero values. If zero, no grid lines will be drawn.
L,M	these integers cause every L^{th} vertical and M^{th} horizontal grid line to be redrawn for emphasis. If zero, no grid lines will be emphasized. If negative, a square grid will be produced.
I,J	these integers cause every I^{th} vertical and J^{th} horizontal grid line to be labeled with its value. If zero, no grid lines will be labeled. If negative, the labels will be placed outside the grid, otherwise they will appear on the zero axis.

SC-4060 PLOT SYMBOL DICTIONARY

(for use with quick plot subroutines only)

Integer	Symbol	Integer	Symbol	Integer	Symbol	Integer	Symbol
1	A	31	4	61	m	105	∫
2	B	32	5	62	n	106	∂
3	C	33	6	63	o	107	o
4	D	34	7	64	p	108	<
5	E	35	8	65	q	109	#
6	F	36	9	66	r	110	¬(logical inverse)
7	G	37	(blank)	67	s	111	|
8	H	38	.	68	t	112	π
9	I	39	,	69	u	113	_
10	J	40	'(close quote)	70	v	114	□
11	K	41	$	71	w	115	Σ
12	L	42	(	72	x	116	~(tilde)
13	M	43	)	73	y	117	⊏ (lozenge)
14	N	44	/	74	z	118	Δ
15	Ø	45	-(minus)	88	"	121	←
16	P	46	+	89	¢	122	→
17	Q	47	*	90	[	123	o(circle)
18	R	48	=	91	]	124	·
19	S	49	a	92	?	125	•
20	T	50	b	93	-(hyphen)	126	•
21	U	51	c	94	!	127	•
22	V	52	d	95	;	136	'(open quote)
23	W	53	e	96	:	138	{
24	X	54	f	97	α	139	}
25	Y	55	g	98	β	140	\
26	Z	56	h	99	^ (caret)	141	-(bar)
27	0	57	i	100	δ	142	±
28	1	58	j	102	%	143	@
29	2	59	k	103	γ	144	&
30	3	60	l	104	>		

READ AND REWIND

SUBROUTINE NAMES: READ or WRITE

PURPOSE:

These subroutines enable the user to read and write arrays of data as binary information on magnetic tape. The first argument L must be the integer number of the logical tape being addressed. The second argument X must address the first data value of the array to be written out or starting location for data to be read into. The third argument N must be an integer. For WRITE it is the number of data values to be written on tape as a record. For READ it is the number of data values to be read in from tape from the next record, not necessarily the entire record.

RESTRICTIONS:

The user should check Appendix E to determine which logical tapes are available and control card requirements. All processed information must be in binary.

CALLING SEQUENCE: READ(L,X(DV),N)

or WRITE(L,X(DV),N)

SUBROUTINE NAME: EØF or REWIND

PURPOSE:

These subroutines enable the user to write end of file marks on magnetic tape and to rewind them. They are generally used in conjunction with subroutines READ and WRITE discussed above. The single argument L must be the integer logical tape number of the unit being activiated.

RESTRICTIONS:

The user should check Appendix E to determine available logical tapes.

CALLING SEQUENCE:

EØF(L)

or REWIND(L)

MATRIX PRINTOUT

SUBROUTINE NAME: LIST

PURPOSE:

This subroutine prints the elements of a matrix [A] and identifies each by its row and column number. The user must supply an alphanumeric name ALP and integer number NUM to identify the matrix. This is to maintain consistency with subroutines FILE and CALL.

RESTRICTIONS:

The matrix must have its integer number of rows and columns as the first two data values.

CALLING SEQUENCE:

LIST(A(IC),ALP,NUM)

SUBROUTINE NAME: PUNCH

PURPOSE:

This subroutine punchs out a matrix [A] , size n*m, one column at a time in any desired format. The argument FØR must reference a FØRTRAN format statement that has been input as a positive array. It must include the outer parenthesis but not the word FØRMAT. The argument HEAD must be a single BCD word used to identify the matrix. Each column is designated and restarts use of the FØRMAT statement.

RESTRICTIONS:

The matrix [A] must have exactly enough space and contain the integer number of rows and columns as the first two data values.

CALLING SEQUENCE:

PUNCH(A(IC),HEAD,FØR(IC))

NOTE: This subroutine requires n+3 dynamic storage locations.

SUBROUTINE NAME: SYMLST

PURPOSE:

To print out and identify the element values of a half symmetric matrix This output subroutine is most generally used with subroutine SCRPFA, see page A.8-10.

RESTRICTIONS:

This subroutine has no error checks built in so please exercise caution when using.

CALLING SEQUENCE: SYMLST(A(DV),N)

where A(DV) addresses the 1,1 element and N is the matrix order.

OUTPUT FOR ABLATS

SUBROUTINE NAME: PNTABL

PURPOSE:

To provide output information for users of subroutine ABLATS. The ABLATS routine performs ablative simulation calculations but since it is called in Variables 2 it performs no output. The user must call PNTABL in the Output Calls block and reference the ablative array of the ABLATS call. When the ablative material is expended, ABLATS will call PNTABL directly and also cause current problem time to be printed.

RESTRICTIONS:

This routine is called in conjunction with subroutine ABLATS only, see page A.8-5.

CALLING SEQUENCE: PNTABL(AA(IC))

A.8 APPLICATION SUBROUTINES

Fluid Flow

Phase Change

Thermal Radiation Exchange

FLUID FLOW

SUBROUTINE NAMES: PRESS or SPRESS

PURPOSE:

These routines are useful for impressing nodal pressures in one dimensional flow paths once the entry pressure P1, path conductance G and flow rate W are known. The respective equations are:

$$P2 = P1 - W/G$$

or

$$P1(i+1) = P1(i) - W/G(i), \quad i = 1,2,3,\ldots,N$$

RESTRICTIONS:

For SPRESS, the pressures and conductors must be sequential and in ascending order, the number of pressure points to be calculated must be supplied as the integer N.

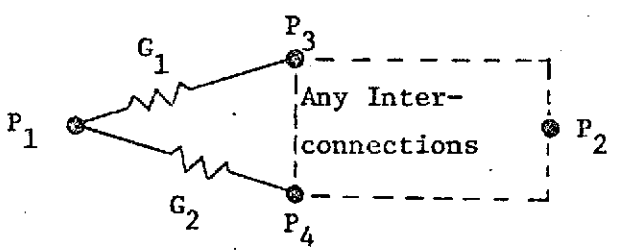
CALLING SEQUENCE: PRESS(P1,W,G,P2)

SPRESS(N,P1(DV),W,G(DV))

SUBROUTINE NAME: EFFG

PURPOSE:

For a pressure network of the following type:



where the values of the identified elements are known, this subroutine will calculate the effective conductance GE from P1 to P2. Any interconnections may occur in the space but only P2, P3 and P4 may be on the boundary and no elements may cross it. The equation utilized is:

$$GE = (G1*(P1-P3) + G2*(P1-P4))/(P1-P2)$$

RESTRICTIONS:

See above. May not be used where capacitors appear on the internal nodes.

CALLING SEQUENCE: EFFG(P1,P2,P3,P4,G1,G2,GE)

FLUID FLOW

SUBROUTINE NAMES: QMETER or RDTNQS or QMTRI or QFØRCE

PURPOSE:

These subroutines are generally used for calculating flow rates. Their respective operations are:

```
          A  =  B*(C-D)
or        A  =  B*((C+460.)**4-(D+460.)**4)
or        Ai =  Bi*(Ci-Ci+1)    ,   i = 1,N
or        Ai =  Bi*(Ci-Di)      ,   i = 1,N
```

RESTRICTIONS:

All values must be floating point numbers except the array length N which must be an integer.

CALLING SEQUENCE:

```
          QMETER(C,D,B,A)
or        RDTNQS(D,C,B,A)
or        QMTRI(N,C(DV),B(DV),A(DV))
or        QFØRCE(N,C(DV),D(DV),B(DV),A(DV))
```

SUBROUTINE NAMES: QINTEG or QINTGI

PURPOSE:

These subroutines perform a simple integration. They are useful for obtaining the integrals of flow rates calculated by QMETER, RDTNQS, QMTRI or QFORCE. Their respective operations are:

```
          S  =  S+Q*DT
or        Si =  Si+Qi*DT    ,   i = 1,N
```

RESTRICTIONS:

All values must be floating point numbers except N which must be an integer. Control constant DTIMEU should be used for the step size when doing an integration with respect to time. These subroutines should be called in Variables 2.

CALLING SEQUENCE:

```
          QINTEG(Q,DT,S)
or        QINTGI(N,Q(DV),DT,S(DV))
```

FLUID FLOW

SUBROUTINE NAME: BIVLV

PURPOSE:

This subroutine allows the user to specify the percentage flow rates through two parallel tubes with common end points. One tube must consist of a single flow conductor (G1) while the other tube may consist of one or more sequential flow conductors (G2(I), I = 1,N). The ratio of flow through G1 divided by the total flow may be calculated in any desired manner and must be supplied as the argument W. The conductor values of either one tube or the other are reduced in order to achieve the desired percentage flow rates irregardless of the pressure drop.

RESTRICTIONS:

N must be an integer. G2 must address the first of the sequential conductors in that tube.

CALLING SEQUENCE: BIVLV(N,W,G1,G2(DV)

PHASE CHANGE

SUBROUTINE NAME: ABLATS

PURPOSE:

To provide a simple ablation (sublimation)capability for the SINDA user. The user constructs the 3-D network without considering the ablative. Then in Variables 2 he simulates 1-D ablative attachments by calling ABLATS. ABLATS constructs the 1-D network and solves it by implicit forward-backward differencing (Crank-Nicholson method) using the time step set by the execution subroutine. Separate ablation arrays (AA) must be used for each ABLATS call. Required working space is obtained from unused program common. Several ABLATS calls thereby share unused common. The user must call subroutine PNTABL(AA) in the Output Calls to obtain ablation totals and temperature distribution.

RESTRICTIONS:

ABLATS must be called in Variables 2 and may be used with any execution subroutine. Subroutines DIDEG1, NEWTR4 and INTRFC are called. All units must be consistent. The Fahrenheit system is required. Temperature varying material property arrays must not exceed 60 doublets. Bivariate material properties may be simulated by calling BVSPSA prior to ABLATS. Cross-sectional area is always considered unity. Thermal conductivity, Stefan-Boltzmann constant and density units must agree in area and length units.

CALLING SEQUENCE: ABLATS(AA(IC),R,CP,G,T,C)

where
- C is the capacitance location of the 3-D node attached to.
- T is the temperature location of the 3-D node attached to.
- G is the location of the material thermal conductivity or the starting location (integer count)of a doublet G vs T array.
- CP is the location of the material specific heat or the starting location (integer count) of a doublet C_p vs T array.
- R is the location of the material density or the starting location (integer count) of a doublet R vs T array.
- AA(IC) is the starting location of the ablation array which must be formatted as follows:
- AA(IC)+1 the ablative line number, a user specified identification integer.
- 2 integer number of sublayers (NSL) desired, ABLATS subtracts from this the number of sublayers ablated.
- 3 the initial temperature of the material, ABLATS replaces this with the outer surface temperature, always in degrees F.
- 4 the impressed outer surface heating rate per unit area, radiation rates not included.
- 5 material thickness; this is replaced by the sublayer thickness.
- 6 surface area of the 3-D node attached to, need not be unity.
- 7 ablation temperature, degrees F.
- 8 heat of ablation.
- 9 Stefan-Boltzmann constant in consistent units.
- 10 surface emissivity.
- 11 space "sink" temperature, degrees F.
- 12 SPACE,N,END where N equals NSI + 4.

NOTE: The outer surface radiation loss is integrated over the time step.
This subroutine requires 3(NSI+1) dynamic storage core locations.

PHASE CHANGE

SUBROUTINE NAME: LQSLTR

PURPOSE:

This subroutine accounts for the phase change energy of a melting or solidifying material. The temperature limits for the reaction must be specified (over at least a 1 degree range) and the phase change energy supplied as a constant rate over the range (Btu/°F). The network is constructed to include the capacitance effects of the phase change material. The network solution subroutines are allowed to calculate incorrect answers based on capacitance effects only; a call to LQSLTR in Variables 2 then performs a corrector operation to account for any phase change occurring (reversability allowed) and returns corrected temperatures. The user is required to store the old temperature of the material (in Variables 1) and supply it as an argument to LQSLTR. This subroutine has a "DØ" loop built in and can be applied to several sequential nodes at once.

RESTRICTIONS:

The number of sequential nodes that this subroutine is to be applied to must be supplied as the integer N. All other arguments must be or address data values.

CALLING SEQUENCE: LQSLTR(N,TL,TH,S(DV),C(DV),TØ(DV),TN(DV))

where N is the integer number of nodes to operated on
TL is the low temperature of the range
TH is the high temperature of the range
S(DV) is the first series value of the phase change energy rate
C(DV) is the first series value of the nodal capacitances
TØ(DV) is the first series value of the old temperatures
TN(DV) is the first series value of the new temperatures

PHASE CHANGE

SUBROUTINE NAME: LQDVAP

PURPOSE:

This subroutine allows the user to simulate the addition of liquid to a node. The network data is prepared as though no liquid exists at the node and is solved that way by the network execution subroutine. Then LQDVAP, which must be called Variables 2, corrects the nodal solution in order to account for the liquid. If the nodal temperature exceeds the boiling point of the liquid, it is set to the boiling point.

The excess energy above that required to reach the boiling point is calculated and considered as absorbed through vaporization. If the liquid is completely vaporized the subroutine deletes its operations. The method of solution holds very well for explicit solutions, but may introduce some error when large time steps are used with implicit solutions.

RESTRICTIONS:

This subroutine must be called in Variables 2.

CALLING SEQUENCE: LQDVAP(T,C,A(IC))

where T is the temperature location of the node.
C is the capacitance location of the node.
A + 1 contains the initial liquid weight.
2 contains the liquid specific heat.
3 contains the liquid vaporization temperature.
4 contains the liquid heat of vaporization.
5 receives the liquid vaporization rate (weight/time)
6 receives the liquid vaporization total (total weight)
7 contains the liquid initial temperature.

THERMAL RADIATION EXCHANGE

SUBROUTINE NAMES: IRRADI or IRRADE

PURPOSE:

These subroutines simulate a radiosity network* within a multiple gray diffuse surface enclosure containing a non-absorbing media. The input is identical for both subroutines. However, IRRADE utilizes explicit equations to obtain the solution by relaxation and IRRADI initially performs a symmetric matrix algebra inverse and thereafter obtains the exact solution implicitly by matrix multiplication. The relaxation criteria of IRRADE is internally calculated and severe enough so that both routines generally yield identical results. However, IRRADE should be used when temperature varying emissivities are to be considered and IRRADI should be used when the surface emissivities are constant. Both subroutines solve for the J node radiosity, obtain the net radiant heat flow rates to each surface and return them sequentially in the last array that was initially used to input the surface temperatures. The user need not specify any radiation conductors within the enclosure.

RESTRICTIONS:

The Fahrenheit system is required. The arbitrary number of temperature arguments may be constructed by a preceding BLDARY call. The emissivity, area, temperature-Q and upper half FA arrays must be in corresponding order and of exact length. The first data value of the FA array must be the integer number of surfaces and the second the Stefan-Boltzmann constant in the proper units and then the FA floating point values in row order. The diagonal elements (even if zero) must be included. As many radiosity subroutine calls as desired may be used. However, each call must have unique array arguments. The user should follow the radiosity routine by SCALE, BRKARY or BKARAD to distribute the Q's to the proper source locations.

CALLING SEQUENCE: IRRADI(AA(IC),Aε(IC),AFA(IC),ATQ(IC))

or IRRADE(AA(IC),Aε(IC),AFA(IC),ATQ(IC))

where the arrays are formatted as follows:

AA(IC),A1,A2,A3,A4,..,AN,END
Aε(IC),ε1,ε2,ε3,ε4,...,εN,END
AFA(IC),N,σ,FA(1,1),FA(1,2),FA(1,3),FA(1,4),FA(1,5),..,FA(1,N)
FA(2,2),FA(2,3),FA(2,4),FA(2,5),..,FA(2,N)
. . . . .
FA(N-2,N-2),FA(N-2,N-1),FA(N-2,N)
FA(N-1,N-1),FA(N-1,N)
FA(N,N),END

ATQ(IC),T1,T2,T3,..,TN,END

where FA(1,2) is defined as A(1)*F(1,2). After the subroutine is performed the ATQ array is ATQ(IC),Q1,Q2,Q3,..,QN,END.
Since $FA_1(1,2) \equiv FA_2(2,1)$ only the upper half triangle of the full FA matrix is required. IRRADI inverts this half matrix in its own area, hence approximately 300 surfaces may be considered using SINDA on a 65K core machine.

*"Radiation Analysis by the Network Method," A. K. Oppenheim, Transaction of the ASME, May 1956, pp. 725-735.

THERMAL RADIATION EXCHANGE

SUBROUTINE NAMES: SLRADI or SLRADE

PURPOSE:

These subroutines are very similar to IRRADI and IRRADE but are designed to solve for the solar heating rates within a enclosure. SLRADI inverts a half symmetric matrix in order to obtain implicit solutions, while SLRADE obtains solutions explicity by relaxation. SLRADE should be used when temperature varying solar absorptivities are to be considered. The second data value of the AFA array must be the solar constant in the proper units. The AT array allows the user to input the angle (degrees) between the surface normal and the surface-sun line. The AI array allows the user to input an illumination factor for each surface which is the ratio from zero to one of the unshaded portion of the surface. The solar constant (S), AT and AI values may vary during the transient for both routines. No input surface temperatures are required. The absorbed heating rates are returned sequentially in the AQ array, the user may utilize SCALE, BRKARY or BKARAD to distribute the heating rates to the proper source locations.

RESTRICTIONS:

These routines are independent of the temperature system being used. All of the array arguments must reference the integer count set by the SINDA pre-processor and be of the exact required length. As many calls as desired may be made but each call must have unique array arguments.

CALLING SEQUENCE: SLRADI(AA(IC),Aε(IC),AFA(IC),AT(IC),AI(IC),AQ(IC))

or SLRADE(AA(IC),Aε(IC),AFA(IC),AT(IC),AI(IC),AQ(IC))

SUBROUTINE NAME: EFFEMS

PURPOSE:

This subroutine calculates the effective emissivity E between parallel flat plates by the following equation:

$$E = 1.0/(1.0/E1 + 1.0/E1 - 1.0)$$

where E1 and E2 are the emissivities of the two surfaces under consideration.

RESTRICTIONS:

Arguments must be floating point numbers.

CALLING SEQUENCE: EFFEMS(E1,E2,E)

THERMAL RADIATION EXCHANGE

SUBROUTINE NAME: SCRPFA

PURPOSE:

To obtain the script FA value for radiant transfer within an enclosure. The input arrays are formatted as shown for subrcutines IRRADI and IRRADE. The second data value in the AFA array is used as a final multiplier, if 1.0 the script FA values are returned, if σ then script σ FA values are returned. The script FA values are returned in the ASFA array which is formatted identical to the AFA array and may overlay it.

RESTRICTIONS:

All array arguments must reference the integer count set by the SINDA pre-processor and all arrays must be exactly the required length.

CALLING SEQUENCE: SCRPFA(AA(IC),Aε(IC),AFA(IC),ASFA(IC))

NOTE: Subroutine SYMLST(ASFA(IC)+3,ASFA(IC)+1) may be called to list the matrix values and identify them by row and column number. This routine and the implicit radiosity routine finalize the half symmetric coefficient matrix and call on SYMINV(AFA(IC)+3,AFA(IC)+1) to obtain the symmetric inverse.

B. THERMAL NETWORK CORRECTION PACKAGE

B.1 Introduction

The thermal network correction package consists of a number of subroutines, many of which are internally programmed as part of a larger program subpackage such as STEP which is discussed in Appendix C. These subpackage programs are not totally integrated and must be employed in a stepwise procedure. Major subpackages are denoted Data Comparison and Plotting, Sensitivity Analysis, and Parameter Correction. Detailed operational procedure from test data to a corrected network[1] and theoretical development[2] are reported elsewhere. Major considerations and users instructions are reported here.

B.2 Theoretical Development

Kalman filtering was chosen over other methods because it offered a way to solve some of the problems presented by temperature measurement sparsity, yet retains solution simplicity when the number of measured temperatures in a region is complete. Governing equations are presented for the case of temperature sparsity and for the special condition of complete temperature measurement.

B.2.1 Sparse Temperature Measurements

Consider the heat balance equation

$$\frac{dT_i}{dt} = \frac{Q_i(t)}{C_i} + \sum_{j=1}^{n} \frac{a_{ij}}{C_i}(T_j - T_i) + \sigma \sum_{j=1}^{n} \frac{b_{ij}}{C_i}(T_j^4 - T_i^4) \qquad (B\text{-}1)$$

$$i = 1,2,\ldots,n$$

where:
- T_i is the temperature of the nth node
- t is the time
- a_{ij} is the conductance
- C_i is the capacitance of the ith node
- b_{ij} is the radiation coefficient

For a thermal model that contains n nodes with m nodal temperatures measured, where $m \leq n$, the random noise corrupted

1. Ishimoto, T., Gaski, J. D., Fink, L. C., "Final Report, Development of Digital Computer Program for Thermal Network Correction, Phase II --Program Development, Phase III--Demonstration/Application," September 1970, 11027-6002-RO-000, TRW Systems Group.

2. Ishimoto, T., Pan, H. M., Gaski, J. D., and Stear, E. B., "Final Report, Development of Digital Computer Program for Thermal Network Correction, Phase I--Investigation/Feasibility Study," January 1969, 11027-6001-RP-00, TRW Systems Group.

measurement vector, $\{y^*\}$, is an m by 1 vector whose elements are given by the m noise corrupted measured temperature. This is given by

$$\{y^*\}^T = (T_1^* \; T_2^* \; T_3^* \; \ldots \; T_i^* \; \ldots \; T_m^*) \tag{B-2}$$

where T_i^* = random noise currupted measured temperature for the ith node, i = 1,2,...,m

Sum of model parameters and isothermal nodes is p. The state vector is a p by 1 vector whose elements are the n nodal temperatures and the (p-n) model parameters. The (p-n) parameters are represented by

$$\left(\frac{Q_i}{C_i}\right) , \left(\frac{a_{ij}}{C_i}\right) , \text{ and } \left(\frac{b_{ij}}{C_i}\right) .$$

The state vector is indicated by

$$\{x\}^T = \left(T_1 \; T_2 \; \ldots \; T_n, \frac{Q_i}{C_i} \ldots \frac{a_{ij}}{C_i} \ldots \frac{b_{ij}}{C_i}\right) \tag{B-3}$$

Relationship between the measurement vector and the state vector is given by the following matirx observation equation:

$$\{y^*\} = [M]\{x\} + \{W\} \tag{B-4}$$

In equation (B-4) [M] is the m by p measurement matrix given by

$$[M] = \left[\begin{array}{c:c} I & \\ (m \times m) & 0 \end{array}\right] \tag{B-5}$$

and {W} is the m by 1 random measurement noise vector whose elements are the random noises associated with the m measured temperatures. This is given by

$$\{W\}^T = (W_1 \; W_2 \; \ldots \; W_m) \tag{B-6}$$

Details of the derivation of the Kalman filter may be found in the cited Reference 2, Page B-1; the following summarizes the Kalman filter equations whereby the correction of thermal model parameters can be obtained sequentially.

$$\{y^*\}_t = [M]_t\{x\}_t + \{W\}_t \tag{B-7}$$

$$\{x\}_{t+\Delta t} = [U]_t\{x\}_t \tag{B-8}$$

$$\{\hat{x}\}_t = \{x_a\}_t + [B]_t(\{y^*\}_t - \{y_a\}_t) \tag{B-9}$$

$$\{y_a\}_t = [M]_t\{x_a\}_t \tag{B-10}$$

$$[B]_t = [A]_t[M]_t^T([M]_t[A]_t[M]_t^T + [W]_t)^{-1} \quad \text{(B-11)}$$

$$[J]_t = ([I] - [B]_t[M]_t)\,[A]_t \quad \text{(B-12)}$$

$$\{x_a\}_{t+\Delta t} = [U]_t\{\hat{x}\}_t \quad \text{(B-13)}$$

$$[A]_{t+\Delta t} = [U]_t[J]_t[U]_t^T \quad \text{(B-14)}$$

where $\{y^*\}_t$ = random noise corrupted measurement vector (temperature) obtained at time, t.

$\{x\}_t$ = value of the state vector (unknown parameters) at time, t.

$\{W\}_t$ = random noises associated with the measured data obtained at time, t.

$\{x\}_{t+\Delta t}$ = value of the state vector (unknown parameters) at time, $t+\Delta t$.

$\{M\}_t$ = measurement matrix evaluated at time, t.

$\{\hat{x}\}_t$ = new estimate of the state vector (unknown parameters after processing the measured data obtained at time, t.

$\{x_a\}_t$ = a priori estimate of the state vector (unknown parameters) before processing the measured data obtained at time, t.

$[B]_t$ = measurement weighting matrix evaluated at time, t (the time varying gain).

$[A]_t$ = $E[(\{x\} - \{x_a\})(\{x\} - \{x_a\})^T]$, error covariance matrix for the a priori estimate state vector.

$[J]_t$ = $E[(\{x\} - \{\hat{x}\})(\{x\} - \{\hat{x}\})^T]$, error covariance matrix for the newly estimated state vector.

$[U]_t$ = transition matrix.

Given the correction scheme whereby the Kalman filter equations are used, the following steps are performed:

(1) First obtain an a priori estimate for the state vector $\{x_a\}_t$ and the associated error covariance matrix $[A]_t$;

(2) Calculate the time varying gain $[B]_t$ using the equation (B-11) and the first set of measured data;

(3) Obtain new estimate for the state vector, $\{\hat{x}\}_t$ using equation (B-9) and the first set of measured data;

(4) Calculate the error covariance matrix, $[J]_t$ for the newly estimated $\{\hat{x}\}_t$ using equation (B-12);

(5) Update the newly estimated state vector, $\{\hat{x}\}_t$ with equation (B-13) to obtain the new a priori estimate at time $t+\Delta T$ and calculate its associated error covariance matrix using equation (B-14).

(6) Repeat Steps (2) to (5) using the new a priori estimate for the state vector and its associated error covariance matrix with the second set of measured data.

(7) Repeat above until all the measured data have been processed or until desirable results* are obtained.

Temperature Dependent Parameters

For temperature dependent parameters, the coefficients are considered to be of the form

$$a_{ij} = a^{\circ}_{ij}\ f(T_i, T_j) \tag{B-15}$$

$$b_{ij} = b^{\circ}_{ij}\ g(T_i, T_j) \tag{B-16}$$

Only the constant portion of a_{ij} and b_{ij}, a°_{ij} and b°_{ij}, is to be corrected and the functions $f(T_i, T_j)$ and $g(T_i, T_j)$ are considered to be known.

Using equations (B-15) and (B-16) for the a_{ij}'s and the b_{ij}'s, the heat balance equation for node i can be written as,

$$\frac{dT_i}{dt} = \frac{Q_i}{C_i} + \sum_{j=1}^{n} \left(\frac{a^{\circ}_{ij}}{C_i}\right) f(T_i, T_j)(T_j - T_i) + \sum_{j=1}^{n} \sigma \left(\frac{b^{\circ}_{ij}}{C_i}\right) g(T_i, T_j)(T_j^4 - T_i^4) \tag{B-17}$$

B.2.2 Complete Temperature Measurements

It was indicated above that if all of the nodes are monitored, a very large network can be corrected. This is possible because the governing heat balance equations can be operated singly and timewise sequentially. The Kalman filter is formulated to take advantage of this special temperature measurement situation.

The Kalman filtering equations may be formulated by first arranging the heat balance equation at the ith node such that the known quantities (hard parameters, temperature, and temperature derivatives, if C is hard) are on one side of the equation and the k unknown quantities (soft parameters) are on the other side.

The set of k equations of the ith node plus some random noise associated with the measurement data will yield the following matrix equation:

$$\{y^{*}_i\} = [M_i]\{x_i\} + [W_i] \tag{B-18}$$

where $\{y^{*}_i\}$ represents an artificial measurement vector at the ith node composed of hard parameters and temperature data.

* Desirable results are those results whose variance are smaller than specified values

$[M_i]$ is the artificial measurement matrix that involves the coefficients of these unknown paramters.

$\{x_i\}$ is the state vector formed with the unknown model parameters.

$\{W_i\}$ is the random noise matrix associated with the measurement data.

If the unknown parameters are considered to be constant, the updating matrix, [U], is essentially an identity matrix. With $\{x_i\}$, $\{y_i^*\}$, $[M_i]$, and $[U_i]$ now formulated, the Kalman filtering method is completely identified by assuming a priori information for the unknown parameters.

After the unknown (soft) parameters for node i are determined the procedure is repeated for the jth node with the exception that any parameter of the jth node that was corrected with the ith node solution is set to corrected values and designated as hard for the jth node.

B.3 Operational Procedure for Correcting a Thermal Network

Operational procedure from test data to a corrected network is a multi-step process with the interface between steps requiring special user attention. Some attention was given to integrate or eliminate some of the interfaces but network size and the need for flexibility requires direct user participation. Higher is the degree of automation, less flexible and less general is the resultant network correction program. The overall operational procedure for thermal network correction recognizes the need for user simplicity but was based upon flexibility and generality considerations. A flow diagram with separate program packages and interfaces is shown in Figure B-1; a description of the operational procedure is reported in Reference 1.

B.4 Data Comparison and Plotting

Comparison of test and analytical temperatures for the purpose of isolating those that are out-of-tolerance requires several sub-steps before temperature comparison can begin. Out-of-tolerance criterion is determined from accuracy assessment of analytical temperatures; for the latter, a sensitivity analysis program called STEP offers a way for this assessment. Discussion of STEP and users instructions are presented in Appendix C.

Due to the indeterminate amount of data that have to be processed, the comparison and plotting capability was coded as two separate subroutines, CØMPAR and PLØTMP. These subroutines are coded in such a manner that they may be called in the same

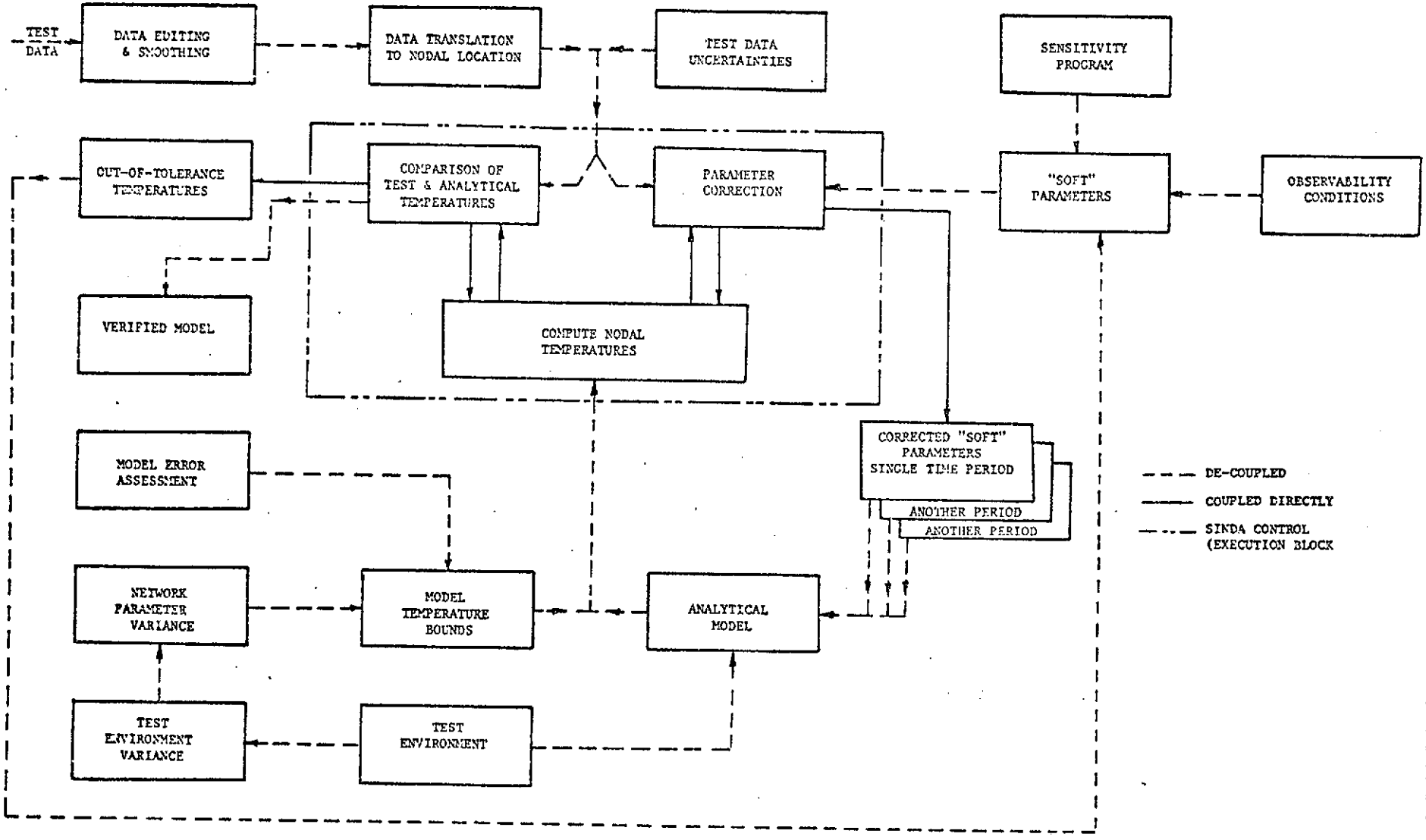


FIGURE B-1 FLOW DIAGRAM - PARAMETER CORRECTION TECHNIQUE

run or in a batched mode. The actual plotting is done by internal calls to SC-4060 quick plot subroutines which have identical names and arguments to the CINDA-3G SC-4060 quick plot subroutines in use at NASA/MSC. Description and users instructions for CØMPAR and PLØTMP are presented in Table B-1 and Table B-2, respectively.

B.5 Parameter Correction

Parameter correction of a large thermal network with temperature sparsity requires a means of assessing unobservability, observability, and the correction of the parameters. Unobservability of a network is determined as part of the KALFIL subroutine and observability of a network is pursued with a separate subroutine called KALØBS. The need for two separate subroutines is a direct result of the two Kalman filtering formulation. Subroutine KALFIL processes the network equations simultaneously whereas subroutine KALØBS processes the network equations singly and sequentially. In general, KALFIL should yield more accurate corrections than KALØBS. Integration of both subroutines into a single package would have unduly complicated the overall thermal network correction package; the user thus must make a decision based upon rather simple ground rules. If a network contains totally measured nodes, KALØBS is used unless the number of nodes plus the number of "soft" parameters total less than 100; for the latter KALFIL is used. If a network contains a region or regions with complete temperature measurements, subroutine KALØBS is called first in order to correct and set hard those "soft" parameters which are totally observable; then subroutine KALFIL is called for the remainder of the network. If a network contains only a limited number of measured temperatures and the measurements are sparsely distributed, subroutine KALFIL is called.

An important consideration that should be discussed here is the accuracy of the "soft" parameter correction. The correction is subject to the observability of the conductors and the accuracy of the measured temperatures. In some instances, the corrected parameter values may be in gross error and physically not realizable, such as a negative conductor, but this should not be particularly surprising since the parameter values merely reflect the accuracy and observability conditions. On the other hand, the calculated temperatures with the corrected parameters should correlate quite closely with the measured temperatures.

B.5.1 Network Correction with Complete Temperature Measurements (KALØBS)

This subroutine is used to correct "soft" parameters that are contained in a totally observable network or subregions. These regions are identified as measured nodes surrounded by measured nodes with the basic smallest totally observable region being a single measured node surrounded by measured nodes. The heat balance equations are processed singly and sequentially with the "soft" parameters set "hard" after correction.

TABLE B-1

TEMPERATURE-TIME HISTORY COMPARISON SUBROUTINE

SUBROUTINE NAME: CØMPAR

PURPOSE:

This subroutine compares two time-temperature history matrices to see if the data sets agree within some specified tolerance. The user must supply an array of integer node numbers in the corresponding order of the temperature data. Those temperature sets which are out-of-tolerance will have the node number set negative in preparation for plotting of out-of-tolerance temperatures by subroutine PLØTMP. (Table B-2).

RESTRICTIONS:

The two time-temperature history matrices must be of equal size and the node numbers input under the indicator array must be in the same order as the matrix temperature data.

CALLING SEQUENCE: CØMPAR(IA(IC),TØL,TM1(IC),TM2(IC))

Where:
- IA is the address of the indicator array
- TØL is the out-of-tolerance criterion (°F)
- TM1 is the first time-temperature matrix array*
- TM2 is the second time-temperature matrix array*

* Refer to page A.6-1 for matrix format (first column is time)

TABLE B-2

TEMPERATURE PLOT SUBROUTINE

SUBROUTINE NAME: PLØTMP

PURPOSE:

This subroutine should be used in conjunction with subroutine CØMPAR. The indicator array is searched until a negative node number is found which indicates an out-of-tolerance condition. The corresponding temperatures from array TM1 and TM2 are then plotted using x and o plotting symbols, respectively. The actual node number from the indicator array is printed as a top line heading. The plot produced requires further processing on the SC-4060.

RESTRICTIONS:

The user should consult Appendix E, Control Cards and Deck Setup, to check tape designation requirements. Subroutines PLØTMP selects the appropriate grid limits and then internally calls upon subroutine PLØTX2. The user must call upon subroutine PLTND after all plotting has been completed.

CALLING SEQUENCE: PLØTMP (IA(IC),TM1(IC),TM2(IC))

Where:	IA	is an indicator array of actual node numbers preprocessed by subroutine CØMPAR
	TM1	is a time-temperature matrix array*
	TM2	is a time-temperature matrix array*

* Refer to page A.6-1 for matrix format (first column is time)

For this subroutine theoretically all (less one) of the parameters associated with a given node may be selected as "soft" and correctable, the user should keep the number of "soft" parameters to a minimum and in general it is better not to mix a "soft" capacitance with "soft" conductances and/or source associated with a given node. User instructions for KALØBS is presented in Table B-3.

Example of an input for subroutine KALØBS is given in Table B-4. The thermal model used for illustrative purposes is the five-node model described in Appendix D. Explanation of the various inputs is indicated directly on the computer print-out as shown in Table B-4. The model considered here has complete temperature measurements, 6 soft conductors and a perturbation factor of 50%. It should be particularly noted in this example that some of the input is for the generation of simulated temperature data and perturbed parameters. The input when experimental data are used would be different from the input shown in the example.

B.5.2 Network Correction with Temperature Sparsity (KALFIL)

This subroutine determines the unobservability of network elements and sets all unobservable elements as "hard" in the indicator vector, thereby eliminating them for corrective consideration. Subregions are identified and dummy pseudo compute sequences formed. These dummy pseudo compute sequences are then utilized by subroutine UMATRX to form the integration matrix utilized in calculating the B and J matrices. (Refer to paragraph B.2.1) Integration of the total network is performed by a standard SINDA network integration subroutine. In this manner the KALFIL parameter correction method for the condition of temperature sparsity is applied to the subregions simultaneously as though the rest of the network was totally hard. A subregion surrounded by unmeasured nodes is less desirable than one surrounded by measured nodes. The latter isolates the subregion from outside influences, while the former is susceptible to error propagation from other subregions not yet corrected. In order to hold external influences to a minimum, all nodes outside the subregions under construction are forced to the measured temperatures, if available.

The conditions of observability and unobservability as determined in Reference 2 are listed in Table B-5. In subroutine KALFIL parameters between unmeasured nodes are automatically set hard (item 6, Table B-5) since these parameters are completely unobservable. For this subroutine, it is better not to mix a "soft" capacitance with "soft" conductances and/or source associated with a given node. Users instructions for KALFIL are presented in Table B-6.

An example of input for subroutine KALFIL is shown in Table B-7, again using the five-node model. This example contains 4 measured temperatures and six soft conductors. Explanation of the various inputs is indicated directly on the computer print-out.

TABLE B-3

KALMAN NETWORK CORRECTION WITH COMPLETE TEMPERATURE MEASUREMENTS

SUBROUTINE NAME: KALØBS

PURPOSE:

This subroutine uses the Kalman filter method to correct soft parameters that are contained in totally observable subregions. This subroutine employs the heat balance equation singly and time wise sequentially. Such a subregion includes all the conductors into a measured node when all the surrounding nodes are also measured. If an adjoining node has identical temperatures as the node under consideration, correction is not possible. If total measurements are not available the user should continue the correction procedure by using the subroutine KALFIL. This routine removes node source and conductor numbers from the IC and IG arrays for corrected parameters. KALØBS is called in the execution block.

RESTRICTIONS:

This subroutine requires the long pseudo-compute sequence (LPCS). The capacitor, source, and conductor indicator arrays must have their contents in the same input order as the node, source and conductor data, respectively. All temperatures must be in the Fahrenheit system.

CALLING SEQUENCE: KALOBS(IPNT,IT(IC),IQ(IC),IC(IC),IG(IC),HT,TNP, QNP,CNP,GNP)

Where:
- IPNT is an intermediate print indicator: I=0,no;I≠0,yes
- IT is an array of actual node numbers of measured temperatures and must be in the same order as the test temperatures
- IQ is an array of actual node numbers of soft sources
- IC is an array of actual node numbers of soft capacitors
- IG is an array of actual conductor numbers of soft conductors
- HT is a time history matrix of test temperatures, each row being a time slice with time as the first value
- TNP is the temperature noise estimate
- QNP is the percent of estimated source error times 0.01
- CNP is the percent of estimated capacitor error times 0.01
- GNP is the percent of estimated conductor error times 0.01

TABLE B-4 EXAMPLE OF INPUT FOR SUBROUTINE KALØBS
FIVE-NODE MODEL, 6 SOFT CONDUCTORS

```
BCD 3THERMAL LPC5
BCD 9 5 NODE PROBLEM FOR CHECKOUT OF KALOBS SUBROUTINE
END
BCD 3NODE DATA
    1,64,9,1,,2,114,6,1,,3,36,4,1,,4,62,1,1,,5,102,6,1,
    -6,,-460,,0,
END
BCD 3SOURCE DATA
SIT 1,A9,1,0
SIT 3,A9,0,5
END
BCD 3CONDUCTOR DATA
SIV 1,1,2,A25,K21,2,1,4,A25,K22,3,1,5,A25,K23
SIV 4,2,3,A25,K24,5,2,5,A25,K25
    6,3,4,,2,7,3,5,,2,8,4,5,,2
GEN -9,4,1,1,0,2,1,,2E-9
    -13,1,6,,2E-9
GEN -14,4,1,2,0,3,1,,2E-9
    -18,3,4,,2E-9,-19,3,5,,2E-9,720,3,6,,2E-9
    -21,4,5,,2E-9,-22,4,6,,2E-9,-23,5,6,,2E-9
END
BCD 3CONSTANTS DATA
    TIMEND,2.0,OUTPUT,0.1
    1=6       $ NUMBER OF SOFT PARAMETERS
    2=6       $ NUMBER OF NODES
    3=0.5          $ PERTURBATION FACTOR
    21,,2,22,,2,23,,2,24,,2,25,,2
END
BCD 3ARRAY DATA
    3,21,6,SPACE,126,END     $ TIME HISTORY MATRIX
    5,SPACE,6,END      $ SPACE FOR INITIAL TEMPERATURES
    9,0.,50.,1.,150.,.2,,50.,,END  $ TIME VARYING Q CURVE, SAWTOOTH
    11,1,2,3,4,5,END  $ MEASURED TEMPS IN ORDER STORED
    12,1,2,3,4,5,6,END      $ NODE NUMBERS FOR PRNTMI
    14,1,4,8,12,18,23,END  $ SOFT CONDUCTOR NUMBERS FOR KALOBS
    15,1,4,8,12,18,23,END  $ SOFT CONDUCTOR NUMBERS FOR PRNTMI
    25,0.,,75,100,,1,25,END      $ VARIABLE CONDUCTIVITY
    91,SPACE,6,END     $ SPACE FOR ORIGINAL PARAMETERS
    92,SPACE,6,END     $ SPACE FOR PERTURBED PARAMETERS
    93,SPACE,6,END     $ SPACE FOR CORRECTED PARAMETERS
    94,SPACE,6,END     $ SPACE FOR ORIGINAL TEMPERATURES
    95,SPACE,6,END     $ SPACE FOR PERTURBED TEMPERATURES
    96,SPACE,6,END     $ SPACE FOR CORRECTED TEMPERATURES

    97,SPACE,6,END     $ SPACE FOR PERCENTAGE OFF
    98
BCD 4     NODE          NUMBER
BCD 4  ORIGINAL         RESULTS
BCD 4 PERTURBED         RESULTS
BCD 4 CORRECTED         RESULTS
    END
    99
```

TABLE B-4 (Cont.)

```
 BCD 4  ORIGINAL      VALUES
 BCD 4 PERTURBED      VALUES
 BCD 4 CORRECTED      VALUES
 BCD 4 CONDUCTOR      NUMBER
 BCD 4 PERCENTAGE      OFF
     END
 END
 BCD 3EXECUTION
DIMENSION X(5000)
NDIM = 5000
NTH = 0
     TPRINT
     GPRINT
     BLOARY(A91,K21,K24,G8,G12,G18,G23) $ SAVE ORIG PARAMETERS
     SHFTV(5,T1,A5)     $ SAVE INITIAL TEMPERATURES
     CNFRDL        $ SIMULATE TEST DATA
     SHFTV(6,T1,A94)          $ SAVE ORIGINAL RESULTS
TIMEO = 0.0
     SHFTV(5,A5,T1)     $ RESET INITIAL TEMPERATURES
THE FOLLOWING SCALE CARD PERTURBS SOFT G FACTORS OR VALUES
     SCALE(K3,K21,K21,K24,K24,G8,G8,G12,G12,G18,G18,G23,G23)
     BLDARY(A92,K21,K24,G8,G12,G18,G23)  $ SAVE PERTURBED PARAMS
     CNFRDL        $ OBTAIN PERTURBED TEMPERATURES
     SHFTV(6,T1,A95)          $ SAVE PERTURBED RESULTS
TIMEO = 0.0
     SHFTV(5,A5,T1)     $ RESET INITIAL TEMPERATURES
     KALORS(0,A11,0.,A14,A3,0.01,0.0,0.0,1.0)
     BLDARY(A93,K21,K24,G8,G12,G18,G23)  $ SAVE CORRECTED PARAMS
     SUBARY(K1,A93,A91,A97)    $ OBTAIN CORRECTION DIFFERENCE
     DIVARY(K1,A97,A91,A97) $ CONVERT TO PERCENT
     ARYMPY(K1,A99,100.0,A97)
     PRNTMI(K1,A99+13,A15+1,A99+1,A91,A99+5,A92,A99+9,A93
            A99+17,A97)  $ PRINT THE CONDUCTOR DATA
     SHFTV(5,A5,T1)     $ RESET INITIAL TEMPERATURES
TIMEO = 0.0
     CNFRDL        $ OBTAIN CORRECTED TEMPERATURES
     SHFTV(6,T1,A96)          $ SAVE CORRECTED RESULTS
     SUBARY(K2,A96,A94,A97)    $ OBTAIN CORRECTION DIFFERENCE
     ARYADD(K2,A94,460.0,A94) $ CONVERT TO RANKINE
     DIVARY(K2,A97,A94,A97) $ CONVERT TO PERCENT, RANKINE BASE
     ARYSUB(K2,A94,460.0,A94) $ CONVERT TO DEGREES F
     ARYMPY(K2,A97,100.0,A97)
     PRNTMI(K2,A98+1,A12+1,A98+5,A94,A98+9,A95,A98+13,A96
            A99+17,A97)  $  PRINT TEMPERATURE DATA
 END
 BCD 3VARIABLES 1
TIMEM = TIMEO
 END
 BCD 3VARIABLES 2
 END
 BCD 3OUTPUT CALLS
     TPRINT

     TESTMP(JTEST,5,T1,TIMEN,A3) $ STORE ANALYTICAL TEMPERATURES
 END
```

TABLE B-5
SUMMARY OF OBSERVABILITY SITUATIONS AND CORRECTIBILITY CONDITIONS
(From Table 2-7, Reference 2)

	SITUATIONS	OBSERVABILITY	ACCURACY
1.	Complete temperature measurements	All parameters are observable	General: Comments discussed below apply
2.	An unmeasured node surrounded by measured nodes	Parameters to unmeasured node are observable	General: Comments discussed below apply
3.	An unmeasured node surrounded by measured nodes and one boundary node	Parameters from measured nodes to the unmeasured node are observable Parameter from boundary node to unmeasured node is unobservable Parameters from the measured node are observable	General: Comments discussed below apply Specific: The correction accuracy is very sensitive to the value of the parameter which is unobservable and thus not correctible
4.	A measured node surrounded by unmeasured nodes	Parameters from the measured node are observable	General: Comments discussed below apply Specific: Even with exact initial temperatures of the unmeasured nodes, the convergence may not be the exact parameter values
5.	A measured node surrounded by unmeasured nodes and one boundary node	Parameters from the measured nodes are observable	The comment for situation 4 applies The value of the parameter from the measured node to the boundary node converges to the true value
6.	Two adjacent unmeasured nodes	Parameters between unmeasured nodes are not observable	Parameters are not correctible
7.	Parallel coupling	Parallel linear coupling is not individually observable Parallel linear and radiation coupling are individually observable	Parallel linear conductors cannot be individually corrected Parallel linear and radiation coupling are individually correctible

General Comment: The accuracy of the parameter correction is dependent upon the accuracy of the experimental temperature data; a quantitative measure is not known at this time. For an unmeasured node the initial temperature must be known accurately; the accuracy of the parameter values are quite sensitive to the initial temperature value.

TABLE B-6

KALMAN NETWORK CORRECTION WITH SPARSE TEMPERATURE MEASUREMENTS

SUBROUTINE NAME: KALFIL

PURPOSE:

This subroutine performs network parameter correction by the Kalman filter method. In general it should be applied to the model being corrected after KALØBS has been applied. This routine must be called upon in the Variables 2 block with CNFRDL in the execution block. It performs an initial pass in order to reduce (set hard) those network elements which are uncorrectable due to observability criteria (unobservable). It then makes a second pass in order to remove from the calculation procedure measured nodes which do not contribute to the solution. It then sets up several square matrices of order N, where N is the number of remaining measured temperatures and soft parameters, and simultaneously solves the Kalman filter set of equations. All corrected parameters are set hard and the corrected values placed into the appropriate network locations. Immediately after the correction process, analytical check runs can be performed.

RESTRICTIONS:

The long pseudo-compute sequence (LPCS) is required. This subroutine must be called within the Variables 2 block by the CNFRDL execution subroutine. Noise or error estimates of zero are not allowed.

CALLING SEQUENCE: KALFIL(I,IT(IC),IC(IC),IQ(IC),IG(IC),AT(IC),AJ(IC))

Where:

- I is an indicator for intermediate printout: I=0,no;I≠0,yes
- IT is an array of actual integer node numbers of the measured temperatures and corresponding to the AT matrix
- IC is an array of actual integer node numbers of soft capacitors and must be in the same order as the node data input
- IQ is an array of actual integer numbers of soft sources and must be in the same order as the source data input
- IG is an array of actual integer conductor numbers of soft conductors; must be in the same order as the conductor data input
- AT is a matrix of test temperature history with the number of rows being the number of time points, the first column representing time and the second column representing test temperatures in the same order as the IT array
- AJ is an array of noise and error estimate squared for each soft parameter and must be in order with IT, IC, IQ and IG

TABLE B-7 EXAMPLE OF INPUT FOR SUBROUTINE KALFIL
FIVE-NODE MODEL, 4 MEASURED TEMPERATURES, 6 SOFT CONDUCTORS

```
BCD 3THERMAL LPCS
BCD 9 5 NODE PROBLEM FOR CHECKOUT OF KALFIL SUBROUTINE
END
BCD 3NODE DATA
    1,64,9,1.,2,114,6,1.,3,36,4,1.,4,62,1,1.,5,102,6,1.
    -6,-460.,0.
END
BCD 3SOURCE DATA
SIT 1,A9,1.0
SIT 3,A9,0.5
END
BCD 3CONDUCTOR DATA
SIV 1,1,2,A25,K21,2,1,4,A25,K22,3,1,5,A25,K23
SIV 4,2,3,A25,K24,5,2,5,A25,K25
    6,3,4,.2,7,3,5,.2,8,4,5,.2
GEN -9,4,1,1,0,2,1,.2E-9
    -13,1,6,.2E-9
GEN -14,4,1,2,0,3,1,.2E-9
    -16,3,4,.2E-9,-19,3,5,.2E-9,-20,3,6,.2E-9
    -21,4,5,.2E-9,-22,4,6,.2E-9,-23,5,6,.2E-9
END
BCD 3CONSTANTS DATA
    1HEND,2,0,OUTPUT,0,1
    1=6        $ NUMBER OF SOFT PARAMETERS
    2=6        $ NUMBER OF NODES
    3=0.5          $ PERTURBATION FACTOR
    21,.2,22,.2,23,.2,24,.2,25,.2
END
BCD 3ARRAY DATA
    2,10,1,0.01,0.01,0.01,0.01          $ NOISE ESTIMATES
      0.01,0.01,0.01,0.1E-19,0.1E-19,0.1E-19,END  $ ERR EST SQURD
    3,21,5,SPACE,105,END    $ TIME HISTORY MATRIX
    5,SPACE,6,END       $ SPACE FOR INITIAL TEMPERATURES
    9,0.,50.,1.,150.,2.,5*.,END  $ TIME VARYING Q CURVE, SAWTOOTH
    11,1,2,3,4,END      $ MEASURED TEMP NODES FOR KALFIL
    12,1,2,3,4,5,6,END      $ NODE NUMBERS FOR PRNTMI
    19,1,4,8,12,18,23,END   $ SOFT CONDUCTOR NUMBERS FOR KALFIL
    15,1,4,8,12,18,23,END   $ SOFT CONDUCTOR NUMBERS FOR PRNTMI
    25,0.,.75,100.,1.25,END      $ VARIABLE CONDUCTIVITY
    91,SPACE,6,END      $ SPACE FOR ORIGINAL PARAMETERS
    92,SPACE,6,END      $ SPACE FOR PERTURBED PARAMETERS
    93,SPACE,6,END      $ SPACE FOR CORRECTED PARAMETERS
    99,SPACE,6,END      $ SPACE FOR ORIGINAL TEMPERATURES
    95,SPACE,6,END      $ SPACE FOR PERTURBED TEMPERATURES
    96,SPACE,6,END      $ SPACE FOR CORRECTED TEMPERATURES
    97,SPACE,6,END      $ SPACE FOR PERCENTAGE OFF
    98
BCD 4     NODE          NUMBER
BCD 4   ORIGINAL        RESULTS
BCD 4 PERTURBED         RESULTS
BCD 4 CORRECTED         RESULTS
    END
```

TABLE B-7 (Cont.)

```
      9
 BCD 4  ORIGINAL       VALUES
 BCD 4 PERTURBED       VALUES
 BCD 4 CORRECTED       VALUES
 BCD 4 CONDUCTOR       NUMBER
 BCD 4 PERCENTAGE       OFF
      END
 END
 BCD 3EXECUTION
DIMENSION X(5000)
NDIM = 5000
NTH = 0
      TPRINT
      GPRINT
      BLDARY(A91,K21,K24,G8,G12,G18,G23) $ SAVE ORIG PARAMETERS
      SHFTV(5,T1,A5)     $ SAVE INITIAL TEMPERATURES
      CNFRDL         $ SIMULATE TEST DATA
      SHFTV(6,T1,A94)          $ SAVE ORIGINAL RESULTS
TIMEO = 0.0
      SHFTV(5,A5,T1)     $ RESET INITIAL TEMPERATURES
THE FOLLOWING SCALE CARD PERTURBS SOFT G FACTORS OR VALUES
      SCALE(K3,K21,K21,K24,K24,G8,G8,G12,G12,G18,G18,G23,G23)
      BLDARY(A92,K21,K24,G8,G12,G18,G23)  $ SAVE PERTURBED PARAMS
      CNFRDL         $ OBTAIN PERTURBED TEMPERATURES
      SHFTV(6,T1,A95)          $ SAVE PERTURBED RESULTS
TIMEO = 0.0
      SHFTV(5,A5,T1)     $ RESET INITIAL TEMPERATURES
ITEST = 1
      CNFRDL         $ PERFORM CORRECTION RUN
      BLDARY(A93,K21,K24,G8,G12,G18,G23)  $ SAVE CORRECTED PARAMS
      SUBARY(K1,A93,A91,A97)    $ OBTAIN CORRECTION DIFFERENCE
      DIVARY(K1,A97,A91,A97) $ CONVERT TO PERCENT
      ARYMPY(K1,A97,100.0,A97)
      PRNTMI(K1,A99+13,A15+1,A99+1,A91,A99+5,A92,A99+9,A93
             A99+17,A97)  $ PRINT THE CONDUCTOR DATA
      SHFTV(5,A5,T1)     $ RESET INITIAL TEMPERATURES
ITEST = 0
TIMEO = 0.0
      CNFRDL         $ OBTAIN CORRECTED TEMPERATURES
      SHFTV(6,T1,A96)          $ SAVE CORRECTED RESULTS
      SUBARY(K2,A96,A94,A97)    $ OBTAIN CORRECTION DIFFERENCE
      ARYADD(K2,A94,460.3,A94) $ CONVERT TO RANKINE
      DIVARY(K2,A97,A94,A97) $ CONVERT TO PERCENT, RANKINE BASE
      ARYSUB(K2,A94,460.3,A94) $ CONVERT TO DEGREES F
      ARYMPY(K2,A97,100.3,A97)
      PRNTMI(K2,A98+1,A12+1,A98+5,A94,A98+9,A95,A96+13,A96
             A99+17,A97)  $  PRINT TEMPERATURE DATA
END
```

TABLE B-7 (Cont.)

```
BCD 3VARIABLES 1
END
BCD 3VARIABLES 2
IF(ITEST.EQ.0) RETURN
      KALFIL(0,A11,0,0,A14,A3,A2)
 END
 BCD 3OUTPUT CALLS
IF(TIMEO.EQ.0.0) CALL VARBL2
      TPRINT
      TESTMP(JTEST,4,T1,TIMEN,A3) $ STORE ANALYTICAL TEMPERATURES
 END
```

B.5.3 Time-Temperature History Matrix (TESTMP)

Subroutine TESTMP which is part of the parameter correction package aids the user in forming a time-temperature history matrix. Users instructions are given in Table B-8.

B.6 Sensitivity Analysis

Accuracy bounds of the analytical temperature may be generated by the use of the sensitivity-temperature error program (STEP). Theoretical development of STEP and brief users instructions are presented in Appendix C. For details on the overall program instructions, the reader should refer to Reference 3. STEP provides a means of generating temperature uncertainty due to parameter uncertainties and a means of assessing the relative "hardness" of "softness" of a parameter with respect to a given temperature.

TABLE B-8

SUBROUTINE NAME: TESTMP

PURPOSE:

This subroutine aids the user in forming a time-temperature history matrix.

RESTRICTIONS:

See below.

CALLING SEQUENCE: TESTMP (I,J,AT(DV),X,AM(IC)

Where:

I	is always a zero integer
J	is the number of values to be stored from AT
AT	is the start of an array of values to be stored in AM
X	is generally TIMEN and is always stored ahead of AT
AM	is a matrix array which must have J+1 columns

NOTE:

This subroutine is generally called upon in the Output Calls block. Each time it is called I is updated by one and another row added to the AM matrix. When AM is full its operation ceases.

C. STEP (Sensitivity Temperature Error Program)

C.1 Introduction

Subroutine STEP (Sensitivity-Temperature Error-Program) generates static sensitivity coefficients that may be used to assess the relative parameter effects on a specified temperature or to assess the uncertainty of a given temperature. Theoretical development is reported elsewhere,* but briefly STEP is based upon a derivative operation on the steady state heat balance equations.

$$Q_i = \sum_{j=1}^{p} a_{ij} (T_i - T_j) + \sum_{j=1}^{p} \sigma b_{ij} (T_i^4 - T_j^4) \qquad \text{(C-1)}$$

$$i = 1,2, \ldots, n$$

where: Q_i is the net heat input to the ith node

a_{ij} is coefficient for conduction and/or convection

b_{ij} is the coefficient for radiation exchange

p is the sum of n variable and p-n fixed temperatures

The derivative operation is conducted in terms of Q_i, a_{ij}, b_{ij}, and T_j ($j > n$) and is expressed in a matrix form; the solution of matrix equations yields the sensitivity coefficients

$$\frac{\partial T_i}{\partial a_{k\ell}}, \text{ for } k = 1, \ldots, n \qquad \text{(C-2)}$$
$$\ell = k+1, \ldots, p$$

$$\frac{\partial T_i}{\partial b_{k\ell}}, \text{ for } i = 1, \ldots, n \qquad \text{(C-3)}$$
$$k = 1, \ldots, n$$
$$\ell = k+1, \ldots, p$$

$$\frac{\partial T_i}{\partial a_{k\ell}}, \text{ for } i = 1, \ldots, n \qquad \text{(C-4)}$$
$$k = 1, \ldots, n$$
$$\ell = k+1, \ldots, p$$

$$\frac{\partial T_i}{\partial Q_k}, \text{ for } i = 1, \ldots, n \qquad \text{(C-5)}$$
$$k = 1, \ldots, n$$

$$\frac{\partial T_i}{\partial T_k}, \text{ for } i = 1, \ldots, n \qquad \text{(C-6)}$$
$$k = n+1, \ldots, p$$

* Ishimoto, T. and Bevans, J. T., "Temperature Variance in Spacecraft Thermal Analysis," J. of Spacecraft, Vol. 3, No. 11, pp 1372-1376, November 1968.

The sensitivity coefficients indicated by equations C-2 through C-6 are employed to generate temperature deviation expressions.

Random Temperature Deviation

The temperature deviation in the random sense, $(\Delta T_i)_r$ may be expressed as:

$$(\Delta T_i)_r = \left\{ \sum_{k=1}^{n} \left(\frac{\partial T_i}{\partial Q_k} \Delta Q_k\right)^2 + \sum_{k=n+1}^{p} \left(\frac{\partial T_i}{\partial T_k} \Delta T_k\right)^2 + \sum_{k=1}^{n} \sum_{\ell=k+1}^{p} \left[\left(\frac{\partial T_i}{\partial a_k} \Delta a_{k\ell}\right)^2 + \left(\frac{\partial T_i}{\partial b_{k\ell}} \Delta b_{k\ell}\right)^2 \right] \right\}^{1/2} \quad \text{(C-7)}$$

Linear Algebraic Temperature Deviation

If the parameter perturbations are deterministic, then the temperature variations should be based upon the algebraic sum of the individual parameter perturbation effects. If $(\Delta T_i)_a$ represents the linear algebraic temperature deviation, the expression is written as:

$$(\Delta T_i)_a = \sum_{k=1}^{n} \frac{\partial T_i}{\partial Q_k} \Delta Q_k + \sum_{k=n+1}^{p} \frac{\partial T_i}{\partial T_k} \Delta T_k + \sum_{k=1}^{n} \sum_{\ell=k+1}^{p} \left(\frac{\partial T_i}{\partial a_{k\ell}} \Delta a_{k\ell} + \frac{\partial T_i}{\partial b_{k\ell}} \Delta b_{k\ell}\right) \quad \text{(C-8)}$$

$$i = 1, 2, \ldots, n$$

Linear Absolute Temperature Deviation

If a worst-case temperature deviation is desired, the partial derivatives and the individual parameter perturbations are evaluated in an "absolute" sense. If $(\Delta T_i)_{ab}$ represents the linear absolute temperature deviation, the expression is written as:

$$(\Delta T_i)_{ab} = \sum_{k=1}^{n} \left| \frac{\partial T_i}{\partial Q_k} \Delta Q_k \right| + \sum_{k=n+1}^{p} \left| \frac{\partial T_i}{\partial T_k} \Delta T_k \right| + \sum_{k=1}^{n} \sum_{\ell=k+1}^{p} \left[\left| \frac{\partial T_i}{\partial a_{k\ell}} \Delta a_{k\ell} \right| + \left| \frac{\partial T_i}{\partial b_{k\ell}} \Delta b_{k\ell} \right| \right] \quad \text{(C-9)}$$

$$i = 1, 2, \ldots, n$$

C.2 STEP User's Directions

C.2.1 SUBROUTINE NAME: STEP

C.2.2 PURPOSE

C.2.2.1 For generating static sensitivity coefficients with respect to $a_{k\ell}$, $b_{k\ell}$, Q_k, and T_k (for boundary nodes).

C.2.2.2 For generating temperature deviation in the root mean square sense, in the algebraic sum sense, and in the absolute value sense.

C.2.3 RESTRICTIONS

C.2.3.1 CINDSL must be called before STEP is called since the long pseudo compute sequence and the arrays containing temperatures, conductances, and heating rates are utilized.

C.2.3.2 Parallel linear or parallel radiation conductors are not permitted.

C.2.3.3 A1 and A2 must be positive arrays.

C.2.3.4 The maximum number of nodes (diffusion plus arithmetic) that can be accommodated is approximately 200.

C.2.4 CALLING SEQUENCE

C.2.4.1 STEP(A1(IC),A2(IC))

where: A1(IC) is the array number for print specifications

A2(IC) is the array number for variance specifications

C.2.4.2 Format, A1 and A2

IC,PC,Op,Op,...,Op,PC,Op,...,END

where: IC is the array number

PC is a parameter code (Refer to Table C-1)

Op is an option (for A1, refer to Table C-2; for A2 refer to Table C-3).

C.2.5 NOTES

C.2.5.1 This subroutine requires $N^2 + P$ locations of dynamic storage. N is the sum of diffusion and arithmetic nodes (non boundary nodes) and P is the total number of nodes (diffusion plus boundary nodes).

C.2.5.2 In Table C-2, for the option designated by NODE, approximately 100 node numbers may be specified.

C.2.5.3 In Table C-3, approximately 35 individual variances may be specified, not including those generated under option ALL.

C.2.5.4 Based upon a number of models ranging from 30 to 164 nodes, the solution time can be estimated in a very approximate sense by,

$$\text{Solution time (minutes)} = 1/2 \left(\frac{n}{30}\right)^2$$

where n is the number of nodes (diffusion plus arithmetic)

C.2.6 ILLUSTRATIVE STEP INPUT

The STEP input itself is illustrated directly below, but it should be noted that STEP requires SINDA input considerations. The combined STEP-SINDA requirements are illustrated in a step by step fashion in Table C-4.

C.2.6.1 Array A1

```
1,A,LIST,ALL
  B,PURE,ALGORD,NODE,1,4
  Q,ABSORD,NODE,2,3
  CONT,ALL,PURE,DELTA
  DELTAT,END
```

C.2.6.2 Array A2

```
2,A,ALL,.1,B,ALL,.1,3,4,.05
  Q,ALL,.08,2,.1,4,.1,CONT,ALL,.05,END
```

C.2.6.3 Note that the node numbers specified are actual, not relative numbers.

C.2.7 FLOW DIAGRAM

A flow diagram of the major logic for the STEP subroutine is presented in Figure C-1.

TABLE C-1

PARAMETER CODE	PARAMETER
A	Linear conductors
B	Radiation conductors
Q	Source terms (heating rates)
CONT	Constant temperatures
DELTAT	This is not a parameter, but rather a signal to the program to calculate and print the three types of deviation. Use in array A1 only. Options in Table C-2 do not apply.

TABLE C-2

OPTIONS	EXPLANATION OF OPTIONS
LIST	The parameters and variance printed.
ALL	All sensitivity coefficients multiplied by the parameter variance are printed.
PURE	Sensitivity coefficients (not multiplied by parameter variance) are printed.
DELTA	Sensitivity coefficients multiplied by parameter variance are printed; this option need be used only in conjunction with option PURE.
PURE, DELTA	Both outputs under PURE and DELTA are printed.
ALGORD	Each set of output called by ALL or PURE is arranged by the magnitude of algebraic values from the largest to the smallest.
ABSORD	Each set of output called by ALL or PURE is arranged by the magnitude of absolute values from the largest to the smallest.
MULT,n	Output as called by ALL or PURE is limited by the print limiting multiplier, n. The sensitivity coefficient, $\frac{dT}{dP}$, is not printed if $\left\|\frac{dT}{dP}\right\| < n \left\|\frac{dT}{dP}\right\|_{max}$
NODE,i_1,i_2,...	The sensitivity coefficients for the specified nodes (i_1,i_2,...) are printed.

TABLE C-3

OPTIONS	EXPLANATION OF OPTIONS
ALL,n,i_1,m_1,i_2,m_2,..,i_k,m_k	This option applies to parameter codes Q and CONT only. The variance will be computed from the relationship, variance of parameters = n (value of parameter) unless exceptions denoted by i_1, m_1, i_2, m_2,..., i_k, m_k are specified, i_k is the node number and m_k is the factor defined in the same manner as n.
ALL,n,i_1,j_1,m_1,...	This option applies to parameter codes A and B only. It is similar to the option above except that the user must supply the adjoining node numbers (i_k,j_k) for the conductors.

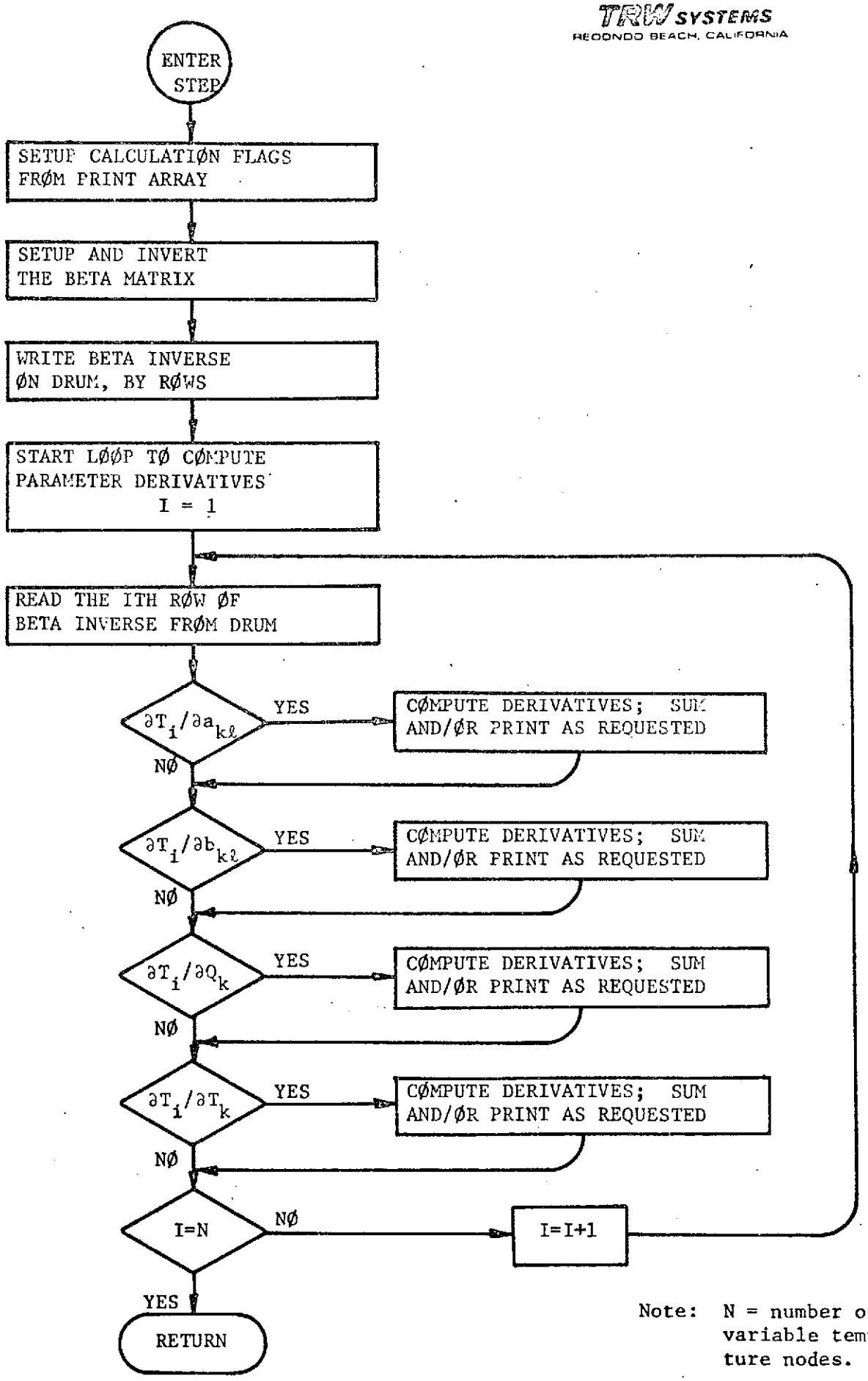
TABLE C-4 ILLUSTRATION OF COMBINED SINDA-STEP INPUT

ITEM	INPUT	COMMENTS
(1)	BCD 3THERMAL LPCS	
(2)	BCD 9 TITLE	Optional
	" "	"
(3)	END	
(4)	BCD 3NODE DATA	(Refer to Section 4.2.2)
(4.1)	N#, T_i, C	N# is the actual node number Nodes with a capacity value are identified as diffusion nodes.
(4.2)	N#, T_i, -1	-1 means arithmetic node
(4.3)	-N#, T_i, 0	The minus sign in front of N# means boundary node. 0 is a convenient number to use as a space holder in core (do not leave blank)
(5)	END	Note: A table relating the actual node number to the relative node number will be automatically printed.
(6)	BCD 3CONDUCTOR DATA	(Refer to Section 4.2.4)
(6.1)	G#, NA, NB, G	G# is the relative conductor number. G is the conductor value between adjoining node number NA & NB
(6.2)	-G#, NA, NB, G	The minus sign in front of G# means radiation coupling. G means the radiation coefficient value.
		Note: A table relating the actual conductor number to the relative conductor number will be automatically printed.
(7)	END	

ITEM	INPUT	COMMENTS
(8)	BCD 3CONSTANTS DATA	(Refer to Section 4.2.5)
(8.1)	NLOOP,N_1,DRLXCA,N_2,ARLXCA,N_3,DAMPA,N_4, DAMPD,N_5	NLOOP means the number of iterations; typically N_1 = 200 to 1000. DRLXCA means diffusion nodes relaxation criterion allowed (convergence criterion for diffusion nodes); typically, N_2 = .001 to .01. DAMPA means damping for arithmetic nodes; typically, N_4 = .7 but for radiation dominated problems, N_4 may be smaller than .001. DAMPD means damping for diffusion nodes; typically, same as DAMPA. Note: $T_{new} = (1 - N_4)\ T_{old} + N_4\ T'_{new}$
(8.2)	K_1, N_1, K_2, N_2,..., K_n, N_n	K_n represents a number used in conjunction with BCD 3 variables, which is discussed below (Item 14). N_n represents heating values.
(9)	END	
(10)	BCD 3ARRAY DATA	The array data contain the input data for the sensitivity coefficients of each parameter category denoted as A, B, Q, or CONT. The input designations and explanation are presented in Tables C-1, C-2, and C-3. To illustrate the options that are offered, the STEP input example of Section C.2.6 will be presented and discussed.
(10.1)	1, A, LIST, ALL	1 means array 1. A means linear conductors. LIST means that the A parameters and variance will be printed. ALL means that all sensitivity coefficients, in terms of A, multiplied by the parameter variance will be printed.

ITEM	INPUT	COMMENTS
(10.11)	B, PURE, ALGORD, 1, 4	B means radiation conductors. PURE, ALGORD, NODE, 1, 4 means that the sensitivity coefficients in terms of B, but not multiplied by the variance, will be printed for nodes 1 and 4 only and will be arranged from the largest to the smallest in accordance with the algebraic values. Note: If all of the sensitivity coefficients were desired, the input would read: B, PURE, ALL, ALGORD.
(10.12)	Q, ABSORD NODE, 2, 3	Q means heating rates. ABSORD, NODE, 2, 3 means that the sensitivity coefficients (multiplied by the variance) of nodes 2 and 3 will be printed and arranged from the largest to the smallest in accordance with the absolute values.
(10.13)	CONT, ALL, PURE, DELTA	CONT, ALL, PURE, DELTA means that both of the outputs, as called by options PURE and DELTA, will be printed in terms of boundary temperatures, CONT.
(10.14)	DELTAT, END	DELTAT means that the three types of temperature deviation will be calculated and printed. END means the end of data.
(10.2) (10.21)	2, A, ALL, .1, B, ALL, .1, 3, 4, .05, Q, ALL, .08, 2, .1, 4, .1, CONT ALL, .05, END	This input means that all variance of A is equal to .1A, all variance of B is .1B except for B between nodes 3 and 4, which is equal to .05B, all variance of Q is equal to .08Q, except for node 2, which is equal to .1Q and node 4 which is equal to .1Q, and all variance of CONT (boundary temperatures) which is equal to .05CONT.
(11)	END	

ITEM	INPUT	COMMENTS
(12)	BCD 3EXECUTION	
(13)	DIMENSION X(N_c)	
(13.1)	NTH=0	
(13.2)	NDIM = N_c	Dimension X(N_c) represents the amount of core storage required. The number N can be estimated from A.5.1, $N_c = N^2 + P$. N is the sum of diffusion and arithmetic nodes (non-boundary nodes) and P is the total number of nodes including boundary nodes. NTH=0 is a pointer.
(13.3)	ARYMPY (N_R, GN_1, σ, GN_1)	ARYMPY means array multiplier; N_R is the total number of radiation conductions starting from conductor number GN_1; $\sigma = .1714 \times 10^{-8}$.
(13.4)	CINDSL	CINDSL is a steady state netowrk solution subroutine.
(13.5)	STEP (A1, A2)	Calling sequence for STEP.
(14)	END	
(15)	BCD 3 VARIABLES 1 STFSEP (K2,Q2,Q13)	BCD 3VARIABLES 1 is used in conjunction with Item (8.2). STFSEP literally means stuff separately; this means that the heating rate Q_1 on node 1 has the value N_1 of item (8.2). Nodes 2 and 13 have the heating values N_2 of item (8.1), and so forth.
(16)	END	
(17)	BCD 3VARIABLES 2	
(18)	END	
(19)	BCD 3OUTPUT CALLS	
(19.1)	TPRINT	
END		



Note: N = number of variable temperature nodes.

FIGURE C-1. FLOW DIAGRAM OF MAJOR LOGIC

D. EXAMPLE OF SINDA USERS INSTRUCTIONS

D.1 Introduction

Appendix B provided instructions on the use of the various subroutines available in the thermal network correction package and Appendix C provided detailed input procedures on the sensitivity temperature error program (STEP) for the generation of thermal sensitivity coefficients and temperature deviations. Since these instructions were tailored for specific subroutines, many of the options available on SINDA were not used. As a result additional user explanations are presented here.

D.2 Physical System and Mathematical Model

D.2.1 Physical System

The physical system is a hollow "thin-shelled" cube with one face open to space as shown in Figure D-1 (a). A variable external heat load is impressed on surfaces 1 and 3.

D.2.2 Mathematical Model

The five-node mathematical model has been generated to describe the physical system in its environment. The heat load on surfaces 1 and 3 as shown in Figure D-1 (c) is depicted as a saw-tooth and several of the conductive couplings are considered to be temperature dependent with a functional form as shown in Figure D-1 (d). Other interconnections and values are shown in Table D-1.

D.2.3 Objective

The transient temperature response of each of the five nodes is desired.

D.3 Users Instructions

Among the numerous transient network subroutines (Appendix A.2), the user must decide upon a particular numerical integration scheme. For this example, the explicit forward differencing method CNFRDL (A.2.5) with the long pseudo compute sequence (LPCS) will be used. Step-by-step users procedure for this illustrative example follows. The computer listing for this problem is found on pages D-7 through D-9.

D.3.1 Title Block (Refer to Section 4.2.1)

(Col) 8 12

```
BCD 3THERMAL LPCS
BCD 9 5 NODE SAMPLE PROBLEM FOR SINDA
END
```

Comment: Subroutine CNFRDL requires LPCS.

D.3.2 Node Data (Refer to Section 4.2.2)

(Col) 8 12

```
BCD 3NODE DATA
    1,64.9,1.,2,114.6,1.,3,36.4,1.,4,62.1,1.,5,102.6,1.
    -6,-460.,0.
END
```

Comment: Node number, initial temperature, and capacitance; thus 1,64.9,1. means: node number, 1; initial temperature, 64.9°F; and capacitance, 1. Btu/°F. Minus sign in front of node number 6 means boundary node. A dictionary relting relative node number ot the actual node number is given on the computer listing, Page D-7.

D.3.3 Source Data (Refer to Section 4.2.3)

The source data may be inputted in the Source Data Block or in the Variables 1 Operations Block. In this example the Variables 1 Operations Block is employed. In the event the Source Data Block was to be used, the input would require the SIT option for nodes 1 and 3.

D.3.4 Conductor Data (Refer to Section 4.2.4)

(Col) 8 12

```
BCD 3CONDUCTOR DATA
SIV 1,1,2,A25,K21,2,1,4,A25,K22,3,1,5,A25,K23
SIV 4,2,3,A25,K24,5,2,5,A25,K25
    6,3,4,.2,7,3,5,.2,8,5,5,.2
GEN -9,4,1,1,0,2,1,.2E-9
    -13,1,6,.2E-9
GEN -14,4,1,2,0,3,1,.2E-9,1.,1.,1.
    -18,3,4,.2E-9,-19,3,5,.2E-9,-20,3,6,.2E-9
    -21,4,5,.2E-9,-22,4,6,.2F-9,-23,5,6,.2E-9
END
```

Comment: SIV option (refer to Page 4-15) allows linear interpolation of a temperature varying property; input 1,1,2,A25,K21 means: conductor,1; between nodes 1 and 2; temperature varying values in Array 25; multiplied by constant in address K21. If the conductor is a constant a blank code is used; thus, 6,3,4,.2: means conductor 6; between nodes 3 and 4; with value .2.

GEN option (refer to Page 4-9) allows the user to generate a sequence of conductors; thus -9,4,1,1,0,2,1,.2E-9 means: starting with conductor number 9 (minus indicates radiation coefficient), four conductors, 9,10,11, and 12, between nodes 1 & 2, 1 & 3, 1 & 4, and 1 & 5, respectively with a value of .2E-9 will be generated.

A dictionary relating relative conductor number to actual conductor number is given on the computer listing, page D-7.

D.3.5 Constants Data (Refer to Section 4.2.5)

```
(Col)   8   12
        BCD 3CONSTANTS DATA
            TIMEND,2.0,OUTPUT,0.1
            21,.2,22,.2,23,.2,25,.2,25,.2
        END
```

Comment: Control constants are listed in Section 4.2.5.2; TIMEND,2.0 means that the stop time for the transient analysis is 2.0 (hrs). OUTPUT,0.1 means that the interval for activating OUTPUT CALLS is 0.1.

The numbers 21,.1 mean constants address 21 (this is concerned with the conductor data D.3.3) with a value of 0.1.

D.3.6 Array Data (Refer to Section 4.2.6)

```
(Col)   8   12
        BCD 3ARRAY DATA
            9,0.,50.,1.,150.,2.,50.,END  $TIME VARYING Q CURVE,SAWTOOTH
            25,0.0,0.75,100.,1.25,END    $VARIABLE CONDUCTIVITY
        END
```

Comment: Heat input at three different time points (0., 1., & 2.) are stored in array 9. At points between the data, a linear interpolation is used. Array 25 contains the thermal conductivity value at two different temperature points, 0°F and 100°F.

D.3.7 Execution Operations (Refer to Section 4.2.7.2)

```
(Col)   8   12
        BCD 3EXECUTION
       DIMENSION X(5000)
       NDIM = 5000
       NTH = 0
            CNFRDL
        END
```

Comment: 5000 represents the working location. CNFRDL is the explicit forward differencing subroutine (Appendix A.2.5).

D.3.8 Variables 1 Operations (Refer to Section 4.2.7.3)

```
(Col)   8   12
        BCD 3VARIABLES 1
            D1DEG1(TIMEN,A9,Q1)   $ TIME VARYING Q ON NODE 1
            MLTPLY(Q1,0.5,Q3)     $ VARIABLE Q ON NODE 3
        END
```

Comment: D1DEG1 is the single variable linear interpolation subroutine (Page A.4-4).

D.3.9 Variables 2 Operations (Refer to Section 4.2.7.4)

(Col) 8 12

```
BCD 3VARIABLES 2
END
```

Comment: No variables 2 operations required.

D.3.10 Output Calls (Refer to Section 4.2.7.5)

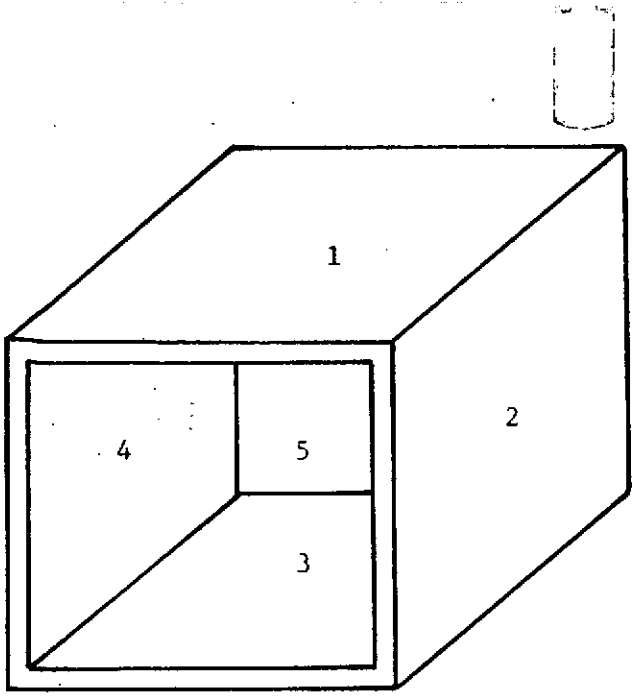
(Col) 8 12

```
BCD 3OUTPUT CALLS
    TPRINT
END
```

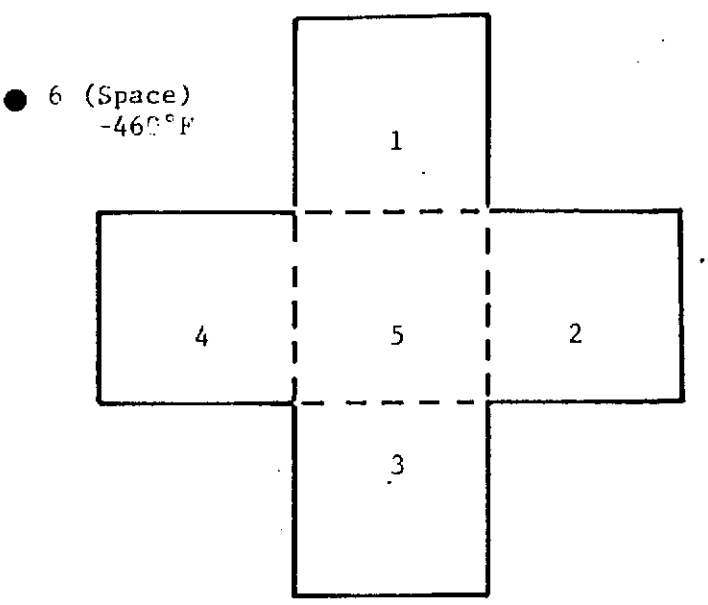
Comment: TPRINT is the output call for all nodal temperatures (Page A.7.3).

D.4 Computer Listing

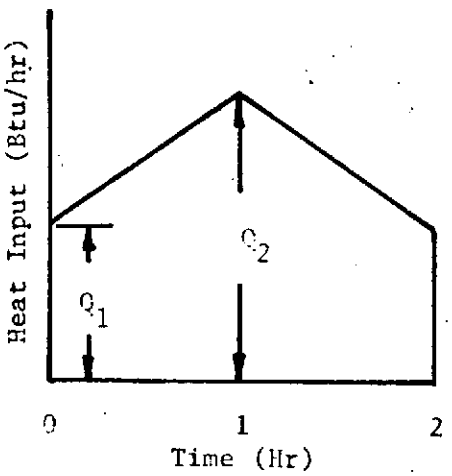
The computer listing for this five-node example is found on Pages D-7 through D-9.



(a) Five-Node Model



(b) Five-Node Model Folded Inside Out



HEAT INPUT

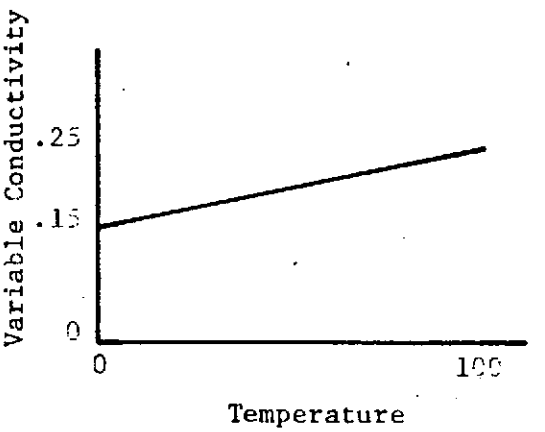
Variable

Node 1, Q_1 = 50, Q_2 = 150

Node 3, Q_1 = 25, Q_2 = 75

Nodes 2,4,5 Q = 0

(c) Heat Input Conditions



Conductors: a_{12}

a_{14}

a_{15}

a_{23}

a_{25}

(d) Variable Conductors

FIGURE D-1. FIVE-NODE HOLLOW CUBE MODEL

TABLE D-1

NOMINAL PARAMETER VALUES, FIVE-NODE MODEL, VARIABLE HEAT INPUT

Coefficients	Value	Coefficients	Value	Coefficients	Value
a_{12}	Figure D-1	σb_{15}	2.0×10^{-10} (no dimen.)	σb_{56}	2.0×10^{-10} (no dimen.)
a_{14}	"	σb_{23}	2.0×10^{-10} (no dimen.)	C_1	1.0 Btu/°F
a_{15}	"	σb_{24}	2.0×10^{-10} (no dimen.)	C_2	1.0 Btu/°F
a_{23}	"	σb_{25}	2.0×10^{-10} (no dimen.)	C_3	1.0 Btu/°F
a_{25}	"	σb_{26}	2.0×10^{-10} (no dimen.)	C_4	1.0 Btu/°F
a_{34}	0.2 Btu/hr °F	σb_{34}	2.0×10^{-10} (no dimen.)	C_5	1.0 Btu/°F
a_{35}	0.2 Btu/hr °F	σb_{35}	2.0×10^{-10} (no dimen.)	Q_1	Figure D-1
a_{45}	0.2 Btu/hr °F	σb_{45}	2.0×10^{-10} (no dimen.)	Q_2	0 Btu/hr
σb_{12}	2.0×10^{-10} (no dimen.)	σb_{46}	2.0×10^{-10} (no dimen.)	Q_3	Figure D-1
σb_{13}	2.0×10^{-10} (no dimen.)	σb_{16}	2.0×10^{-10} (no dimen.)	Q_4	0 Btu/hr
σb_{14}	2.0×10^{-10} (no dimen.)	σb_{36}	2.0×10^{-10} (no dimen.)	Q_5	0 Btu/hr

```
BCD 3THERMAL LPCS
BCD 9 5 NODE SAMPLE PROBLEM FOR SINDA
END
BCD 3NODE DATA
    1,64.9,1.,2,114.6,1.,3,36.4,1.,4,62.1,1.,5,102.6,1.
    -6,-460.,0.
END

RELATIVE NODE NUMBERS                                   ACTUAL NODE NUMBERS

    1  THRU      6                1       2       3       4       5       6
NODE ANALYSIS...  DIFFUSION =     5,  ARITHMETIC =     0,  BOUNDRY =     1,  TOTAL =     6

BCD 3CONDUCTOR DATA
SIV 1,1,2,A25,K21,2,1,4,A25,K22,3,1,5,A25,K23
SIV 4,2,3,A25,K24,5,2,5,A25,K25
    6,3,4,.2,7,3,5,.2,8,4,5,.2
GEN -9,4,1,1,0,2,1,.2E-9
    -13,1,6,.2E-9
GEN -14,4,1,2,0,3,1,.2E-9
    -18,3,4,.2E-9,-19,3,5,.2E-9,-20,3,6,.2E-9
    -21,4,5,.2E-9,-22,4,6,.2E-9,-23,5,6,.2E-9
END

RELATIVE CONDUCTOR NUMBERS                                      ACTUAL CONDUCTOR NUMBERS

    1  THRU     10               1       2       3       4       5       6       7       8       9      10
   11  THRU     20              11      12      13      14      15      16      17      18      19      20
   21  THRU     23              21      22      23
CONDUCTOR ANALYSIS...  LINEAR =     8,  RADIATION =    15,  TOTAL =    23,  CONNECTIONS =    23

BCD 3CONSTANTS DATA
    TIMEND,2.0,OUTPUT,0.1
    21,.2,22,.2,23,.2,24,.2,25,.2
END
BCD 3ARRAY DATA
    8,0.,50.,1.,150.,2.,50.,END  $ TIME VARYING Q CURVE, SAWTOOTH
    25,0.0,0.75,100.,1.25,END    $ VARIABLE CONDUCTIVITY
END
BCD 3EXECUTION
DIMENSION X(5000)                                                F
NDIM = 5000                                                      F
NTH = 0                                                          F
    CNFRDL        $ EXPLICIT FORWARD DIFFFRENCING USING LPCS
END
BCD 3VARIABLES 1
    D1DEG1(TIMEN,A9,Q1)     $ TIME VARYING Q ON NODE 1
    MLTPLY(Q1,0.5,Q3)       $ VARIABLE Q ON NODE 3 ALSO
END
BCD 3VARIABLES 2
END
BCD 3OUTPUT CALLS
    TPRINT
END
```

TRW SYSTEMS IMPROVED NUMERICAL DIFFERENCING ANALYZER - - - SINDA - - - UNIVAC-1108 FORTRAN-V VERSION

5 NODE SAMPLE PROBLEM FOR SINDA

```
**********
TIME=  0.00000    DTIMEU=  0.00000    CSGMIN(    5)=  6.93368-01 TEMPCC(    2)=  0.00000    RELXCC(    0)=  0.00000

T    1=  6.49000+01  T    2=  1.18600+02  T    3=  3.64000+01  T    4=  6.21000+01  T    5=  1.02600+02  T    6= -4.60000+02

**********
TIME=  1.00000-01 DTIMEU=  1.00000-01 CSGMIN(    5)=  6.93368-01 TEMPCC(    2)=  7.93389+00 RELXCC(    0)=  0.00000

T    1=  7.22062+01  T    2=  1.06667+02  T    3=  4.41177+01  T    4=  6.19418+01  T    5=  9.62652+01  T    6= -4.60000+02

**********
TIME=  2.00000-01 DTIMEU=  1.00000-01 CSGMIN(    5)=  6.97212-01 TEMPCC(    3)=  6.91682+00 RELXCC(    0)=  0.00000

T    1=  7.91025+01  T    2=  1.00156+02  T    3=  5.10385+01  T    4=  6.19866+01  T    5=  9.11003+01  T    6= -4.60000+02

**********
TIME=  3.00000-01 DTIMEU=  1.00000-01 CSGMIN(    5)=  7.01917-01 TEMPCC(    1)=  6.62386+00 RELXCC(    0)=  0.00000

T    1=  8.57264+01  T    2=  9.48578+01  T    3=  5.73295+01  T    4=  6.21353+01  T    5=  8.69490+01  T    6= -4.60000+02

**********
TIME=  4.00000-01 DTIMEU=  1.00000-01 CSGMIN(    5)=  7.05686-01 TEMPCC(    1)=  6.45466+00 RELXCC(    0)=  0.00000

T    1=  9.21811+01  T    2=  9.04036+01  T    3=  6.31495+01  T    4=  6.25229+01  T    5=  8.36528+01  T    6= -4.60000+02

**********
TIME=  5.00000-01 DTIMEU=  1.00000-01 CSGMIN(    5)=  7.07959-01 TEMPCC(    1)=  6.36238+00 RELXCC(    0)=  0.00000

T    1=  9.85434+01  T    2=  8.72585+01  T    3=  6.86150+01  T    4=  6.31209+01  T    5=  8.11589+01  T    6= -4.60000+02

**********
TIME=  6.00000-01 DTIMEU=  1.00000-01 CSGMIN(    5)=  7.08867-01 TEMPCC(    1)=  6.32652+00 RELXCC(    0)=  0.00000

T    1=  1.04870+02  T    2=  8.47157+01  T    3=  7.38258+01  T    4=  6.39367+01  T    5=  7.93136+01  T    6= -4.60000+02

**********
TIME=  7.00000-01 DTIMEU=  1.00000-01 CSGMIN(    5)=  7.08526-01 TEMPCC(    1)=  6.33182+00 RELXCC(    0)=  0.00000

T    1=  1.11201+02  T    2=  8.28895+01  T    3=  7.88624+01  T    4=  6.49791+01  T    5=  7.81087+01  T    6= -4.60000+02

**********
TIME=  8.00000-01 DTIMEU=  1.00000-01 CSGMIN(    5)=  7.07039-01 TEMPCC(    1)=  6.35484+00 RELXCC(    0)=  0.00000

T    1=  1.17566+02  T    2=  8.17107+01  T    3=  8.37934+01  T    4=  6.62538+01  T    5=  7.74683+01  T    6= -4.60000+02

**********
TIME=  9.00000-01 DTIMEU=  1.00000-01 CSGMIN(    5)=  7.04699-01 TEMPCC(    1)=  6.41743+00 RELXCC(    0)=  0.00000

T    1=  1.23984+02  T    2=  8.11229+01  T    3=  8.86739+01  T    4=  6.77661+01  T    5=  7.73463+01  T    6= -4.60000+02

**********
TIME=  1.00000+00 DTIMEU=  1.00000-01 CSGMIN(    5)=  7.01318-01 TEMPCC(    1)=  6.49611+00 RELXCC(    0)=  0.00000

T    1=  1.30480+02  T    2=  8.10690+01  T    3=  9.35401+01  T    4=  6.95207+01  T    5=  7.77018+01  T    6= -4.60000+02
```

TRW SYSTEMS IMPROVED NUMERICAL DIFFERENCING ANALYZER - - - SINDA - - - UNIVAC-1108 FORTRAN-V VERSION

5 NODE SAMPLE PROBLEM FOR SINDA

```
**********
TIME=  1.10000+00 DTIMEU=  1.00000-01 CSGMIN(     5)=  6.98585-01 TEMPCC(     1)=  8.60066+00 RELXCC(     0)=  0.00000

T     1=  1.35080+02  T     2=  8.15060+01  T     3=  9.74560+01  T     4=  7.15220+01  T     5=  7.84883+01  T     6=  -6.00000+02

**********
TIME=  1.20000+00 DTIMEU=  1.00000-01 CSGMIN(     5)=  6.95608-01 TEMPCC(     3)=  3.06863+00 RELXCC(     0)=  0.00000

T     1=  1.38126+02  T     2=  8.22770+01  T     3=  1.00525+02  T     4=  7.36282+01  T     5=  7.95597+01  T     6=  -6.00000+02

**********
TIME=  1.30000+00 DTIMEU=  1.00000-01 CSGMIN(     5)=  6.92819-01 TEMPCC(     3)=  2.33694+00 RELXCC(     0)=  0.00000

T     1=  1.39875+02  T     2=  8.32585+01  T     3=  1.02462+02  T     4=  7.57071+01  T     5=  8.07972+01  T     6=  -6.00000+02

**********
TIME=  1.40000+00 DTIMEU=  1.00000-01 CSGMIN(     5)=  6.90073-01 TEMPCC(     4)=  2.05807+00 RELXCC(     0)=  0.00000

T     1=  1.40538+02  T     2=  8.43352+01  T     3=  1.04555+02  T     4=  7.78051+01  T     5=  8.21049+01  T     6=  -6.00000+02

**********
TIME=  1.50000+00 DTIMEU=  1.00000-01 CSGMIN(     5)=  6.87555-01 TEMPCC(     4)=  1.93080+00 RELXCC(     0)=  0.00000

T     1=  1.40285+02  T     2=  8.50365+01  T     3=  1.05676+02  T     4=  7.97089+01  T     5=  8.34064+01  T     6=  -6.00000+02

**********
TIME=  1.60000+00 DTIMEU=  1.00000-01 CSGMIN(     5)=  6.85339-01 TEMPCC(     4)=  1.77720+00 RELXCC(     0)=  0.00000

T     1=  1.39253+02  T     2=  8.64925+01  T     3=  1.06284+02  T     4=  8.15221+01  T     5=  8.46807+01  T     6=  -6.00000+02

**********
TIME=  1.70000+00 DTIMEU=  1.00000-01 CSGMIN(     5)=  6.83479-01 TEMPCC(     1)=  1.70293+00 RELXCC(     0)=  0.00000

T     1=  1.37550+02  T     2=  8.74516+01  T     3=  1.06927+02  T     4=  8.31032+01  T     5=  8.57606+01  T     6=  -6.00000+02

**********
TIME=  1.80000+00 DTIMEU=  1.00000-01 CSGMIN(     5)=  6.82018-01 TEMPCC(     1)=  2.28861+00 RELXCC(     0)=  0.00000

T     1=  1.35262+02  T     2=  8.82735+01  T     3=  1.06186+02  T     4=  8.44634+01  T     5=  8.67289+01  T     6=  -6.00000+02

**********
TIME=  1.90000+00 DTIMEU=  1.00000-01 CSGMIN(     5)=  6.80976-01 TEMPCC(     1)=  2.80699+00 RELXCC(     0)=  0.00000

T     1=  1.32455+02  T     2=  8.89274+01  T     3=  1.05471+02  T     4=  8.55845+01  T     5=  8.75177+01  T     6=  -6.00000+02

**********
TIME=  2.00000+00 DTIMEU=  1.00000-01 CSGMIN(     5)=  6.80385-01 TEMPCC(     1)=  3.27350+00 RELXCC(     0)=  0.00000

T     1=  1.29181+02  T     2=  8.93898+01  T     3=  1.04430+02  T     4=  8.64565+01  T     5=  8.81054+01  T     6=  -6.00000+02

      END OF DATA
```

E. CONTROL CARDS AND DECK SETUP

The SINDA program, although designed for use on a variety of computers with a minimum of change, is presently operational only on the UNIVAC-1108 EXEC-II system. The system control cards, deck setup for the UNIVAC-1108 for the listed installations are reported here. Included in the presentation are the various disk, drum and tape unit designations and other pertinent information.

UNIVAC-1108 DECK SETUP JACOBI COMPUTATION CENTER, LOS ANGELES

The EXEC-II, CUR and FØTRAN V systems software for the UNIVAC-1108 are well suited for operation of the SINDA program. The two portions of the program, absolute Preprocessor and relocatable Variables, are contained on magnetic tape as files one and two respectively. The user must instruct the operator to mount the tape on drive F. The Δ symbol indicates a seven and eight punch in card column one. The deck setup is as follows:

```
Cols 1     6      12
     Δ     RUN
     ΔRX   ASG    F=SINDA
     ΔRX   ASG    K
     ΔN    XQT    CUR
           IN     F
     ΔN    XQT    SINDA/ABS

                                  ← blank card unless RECALL
                                  ← data deck through END ØF DATA

     ΔN    XQT    CUR
           ERS
           IN     F
           TRI    F
     ΔN    FØR,K  SINDA
     ΔN    FØR,K  EXECTN
     ΔN    FØR,K  VARBL1
     ΔN    FØR,K  VARBL2
     ΔN    FØR,K  ØUTCAL

                                  ← "load and go" subroutines with Δ
                                    FØR cards

     ΔN    XQT    SINDA
     Δ     FIN
```

NOTE: See the next page for tape usage requirements for various options.

It is recommended that the SINDA user acquaint himself with the CUR operating system and the basics of FØRTRAN V, particularly logical IF's.

UNIVAC-1108 TAPE USAGE JACOBI COMPUTATION CENTER, LOS ANGELES

UNIT DESIGNATION	FØRTRAN NUMBER	PROGRAM VARIABLE	FUNCTION
DRUM(M)	15	LUT3	Copy of original problem data.
DRUM(D)	4	LUT1	Data number definitions.
F	9	---	SINDA production tape.
DRUM(I)	12	LB3D	Data tape (original problem and all parameter changes).
K	14	LB4P	Program tape (contains generated Fortran routines; SINDA, EXECTN, VARBL1, VARBL2, ØUTCAL).
DRUM	27	INTERN	Data conversion scratch tape.
R	21	LUT7	Problem recall data tape.*
S	22	STAPE	Problem store data tape.*
Reread	0	KRR	Fortran reread unit.

* These tapes need not be assigned if the particular options are not used. The STØREP option requires assigning and saving tapes 14 and 22. The RECALL options requires assigning and mounting the above tapes on 14 and 21 respectively.

UNIVAC-1108 DECK SETUP NASA HOUSTON

The EXEC-II, CUR and FØRTRAN V systems software for the UNIVAC-1108 are well suited for operation of the SINDA program. The two portions of the program, Preprocessor and Variables, are contained in binary on magnetic tape as files one and two respectively. The user must instruct the operator to mount the tape on drive F. The ∇ symbol indicates a seven and eight punch in the card column. The deck setup is as follows:

```
Col 1      6      12
    ∇     (RUN Card)
    ∇     (MSG Card)
    ∇RX   ASG    F=     (See note below)
    ∇RX   ASG    D,J,K,M
    ∇     XQT    CUR
          IN     F
    ∇N    XQT    SINDA/PREPRØ
                                  ← blank card unless RECALL
                                  ← problem data deck through END ØF DATA
    ∇     XQT    CUR
          ERS
          IN     F
          TRI    F
    ∇N    FØR,K  SINDA
    ∇N    FØR,K  EXECTN
    ∇N    FØR,K  VARBL1
    ∇N    FØR,K  VARBL2
    ∇N    FØR,K  ØUTCAL
                                  ← "load and go" subroutines if any,
                                    with ∇ FØR
    ∇N    XQT    SINDA
    ∇     EØF
```

It is recommended that the SINDA user acquaint himself with the CUR operating system and the basics of FØRTRAN V, in particular, logical IF statements.

The operator instruction ticket accompanying the job must have the SINDA production tape designated as input on F and request D, J, K, M scratch tapes. This job's compatable with all the various 1108 systems at MSC and is required to be run under the FØRTRAN V system.

NOTE: The latest SINDA reel number may be obtained from R.L. Dotts, ES551, X3538.

UNIVAC-1108 TAPE USAGE NASA HOUSTON

UNIT DESIGNATION	FØRTRAN NUMBER	PROGRAM VARIABLE	FUNCTION
DRUM(D)	4	LUT1	Data number definitions.
F	8		SINDA production tape.
DRUM(J)	12	LB3D	Data tape (original problem and all parameter changes).
K	13	LB4P	Program tape (contains generated Fortran routines; SINDA, EXECTN, VARBL1, VARBL2, ØUTCAL).
DRUM(M)	15	LUT3	Copy of original problem data.
DRUM(X)	27	INTERN	Preprocessor scratch.
R	21	LUT7	Problem recall data tape.*
S	22	STAPE	Problem store data tape.*
Reread	30	KRR	Fortran reread unit.

* These tapes need not be assigned if the particular options are not used. The STØREP option requires assigning and saving tapes 13 and 22. The RECALL options requires assigning and mounting the above tapes on 13 and 21 respectively.